FMISD19004 Cloud Computing Technologies

Kęstutis Daugėla

2022-01-12

Contents

Introduction

The underlying concept of cloud computing was introduced way back in 1960s by John McCarthy in his book, "The challenge of the Computer Utility". His opinion was that "computation may someday be organized as a public utility." The rest became history and the majority of the software used now is running in the cloud seamlessly (**?**).

Cloud can solve a lot of problems nowadays - starting with reduced cost, enhanced security, and flexible approach (**?**) up to sustainability (**?**) and accessibility around the world. Continuous Integration and Deployment (CI/CD) is easier than even treating now only the applications, but the whole infrastructure as code. This leads to enhanced productivity and cost optimization (**?**).

Is there anything revolutionary in the cloud offerings today? Definitely, no - people used these capabilities for ages. The only difference is the scale and popularity these days.

Cloud services usually are grouped into three categories:

- SaaS (Software as a service) is a software distribution model in which a cloud provider hosts applications and makes them available to end-users over the internet
- PaaS (Platform as a service) is a complete development and deployment environment in the cloud, with resources that enable you to deliver everything from simple cloud-based apps to sophisticated, cloud-enabled enterprise applications

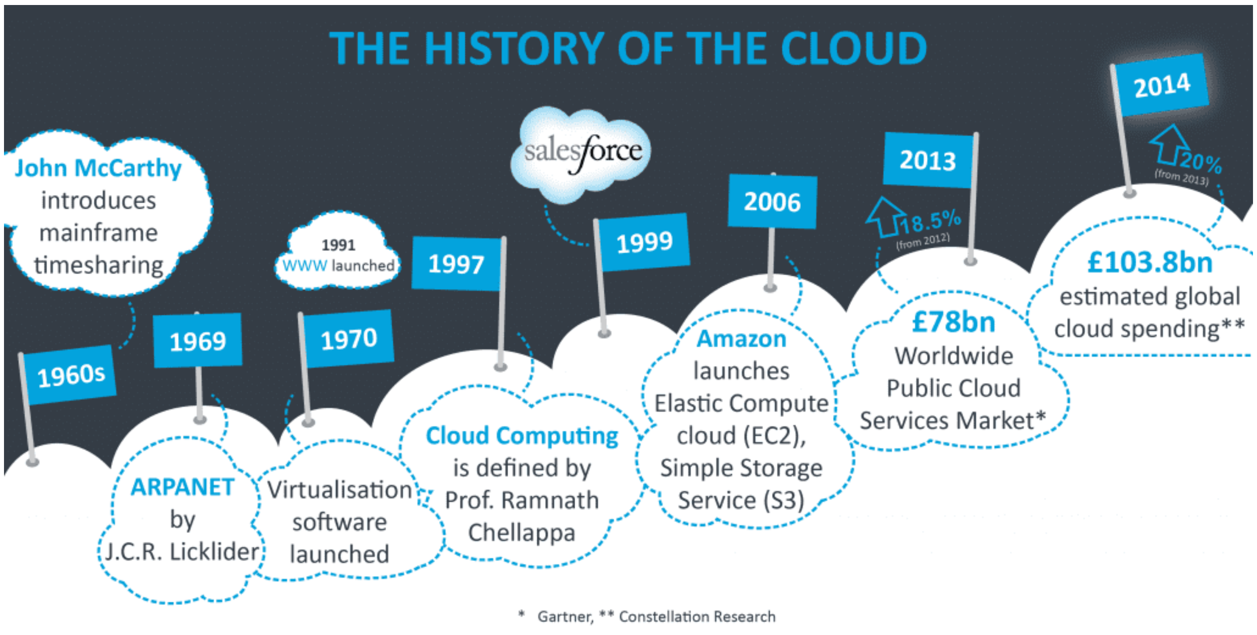

THE HISTORY OF THE CLOUD
John McCarthy introduces mainframe timesharing
1960s
1969
ARPANET by J.C.R. Licklider
1970
Virtualisation software launched
1991 WWW launched
1997
Cloud Computing is defined by Prof. Ramnath Chellappa
salesforce
1999
2006
Amazon launches Elastic Compute cloud (EC2), Simple Storage Service (S3)
2013
18.5% (from 2012)
£78bn Worldwide Public Cloud Services Market*
2014
20% (from 2013)
£103.8bn estimated global cloud spending**
* Gartner, ** Constellation Research

Figure 1: The history of the cloud - image source https://itchronicles.com/

- IaaS (Infrastructure as a service) is a type of cloud computing service that offers essential compute, storage, and networking resources on-demand, on a pay-as-you-go basis

However, despite the gain achieved from cloud computing, organizations are slow in fully accepting it due to security issues and challenges associated with it (**?**).

In terms of the leading cloud service providers, the same three names usually appear - Amazon (AWS), Microsoft (Azure), and Google (GCP). These are also one of the 5 largest companies in the world by market capitalization. While AWS has strength in the engineering supply chain, large financial commitments and innovation, Google demonstrates significant revenue growth, innovation velocity and shows promising results in surveys. Moreover, since Google developed Kubernetes internally, GCP has the most fully-featured Kubernetes service of any provider in this market. Microsoft, on the other hand, already had a good reputation and trust as a software company (**?**).

1 Moving to Cloud

1.1 Managing SLA (SLO) requirements

One of the biggest challenges for potential cloud customers is to evaluate SLA's of cloud vendors (**?**) (**?**). There are four major cloud setups in general:

1) **Public cloud**. In this setup, users can access the resource pool that is managed by a cloud provider. Since this is a public cloud environment, it can pose important security concerns and extra measures need to be taken in order to prevent security issues.
2) **Private cloud**. The vendor provides the services which prevent public assess (e.g. dedicated servers)
3) **Community cloud** The cloud services are provided to a specified group where all members are entitled to equal access to the shared services.
4) **Hybrid cloud** The cloud services are provided as multiple cloud combustion (public cloud, private cloud, and community cloud)

It makes sense to compare the actual numbers between the cloud strategy presented in 2019 with the actual survey made in 2021. Almost every cloud-ready company uses the public cloud (97%) to some extent leaving

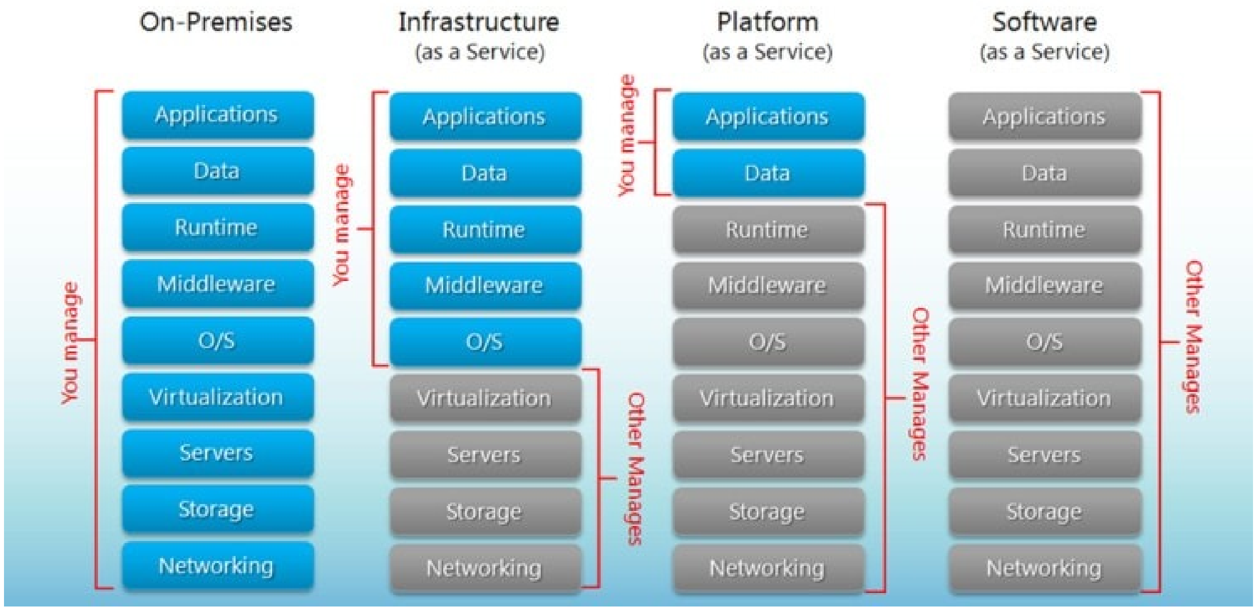


Figure 2: IaaS vs PaaS vs SaaS - image source https://www.bigcommerce.com/blog/saas-vs-paas-vs-iaas/

hybrid cloud setup the dominant one (78%). Companies rarely use public or private cloud alone (19% vs 2% respectively).

When it comes to functionality, cloud providers could cover all of the customers needs either via managed services (e.g. BigQuery, Amazon Redshift, S3, Cloud Pub/Sub) or compute services (e.g. Virtual Machines, Compute engine). While Amazon has the biggest number of internal services and market share, Azure and GCP are chasing the leader rapidly.

Another important factor in choosing a cloud provider is the price. In this area, GCP provides the lowest price for general purpose machines, while compute optimized machines are a little bit cheaper in AWS. Azure provides the best price for memory optimized machines. However, these costs could be optimized significantly by having an agreement with the cloud provider or implementing resource optimizations afterward.

Taking the functional requirements aside, security requirements represent a major issue that has to be met in order of easing some of these obstacles (**?**). Gartner predicts that through 2025, 99% of cloud security failures will be the customer's fault.

Having all these things in mind, it is really hard to draw a conclusion which Cloud provider to choose. At the end of the day, it depends on the specific business requirements, regulations, and personal preference. For instance, if the architecture is heavily based on containers and microservices, GCP could be the best choice since it has the most complete container-based model. The biggest selling point of AWS is that AWS has the greatest Global reach while Azure has more experience in hybrid cloud offerings and Windows-based organization support.

1.2 Migration Approaches

The complexity of migrating existing applications varies, depending on the architecture and existing licensing arrangements. A virtualized, service-oriented architecture can be put on the low-complexity end of the spectrum, and a monolithic mainframe at the high-complexity end of the spectrum (**?**). Cloud computing advocates that resources should be controlled on demand, and can be flexibly and elastically expanded and contracted according to the change of demand (**?**). Therefore it is preferred that applications moving into the cloud must run in a virtualized way, while virtual machines could work as a direct entry for other applications which cannot run directly in the cloud environment (**?**). Automation is an important aspect of migration

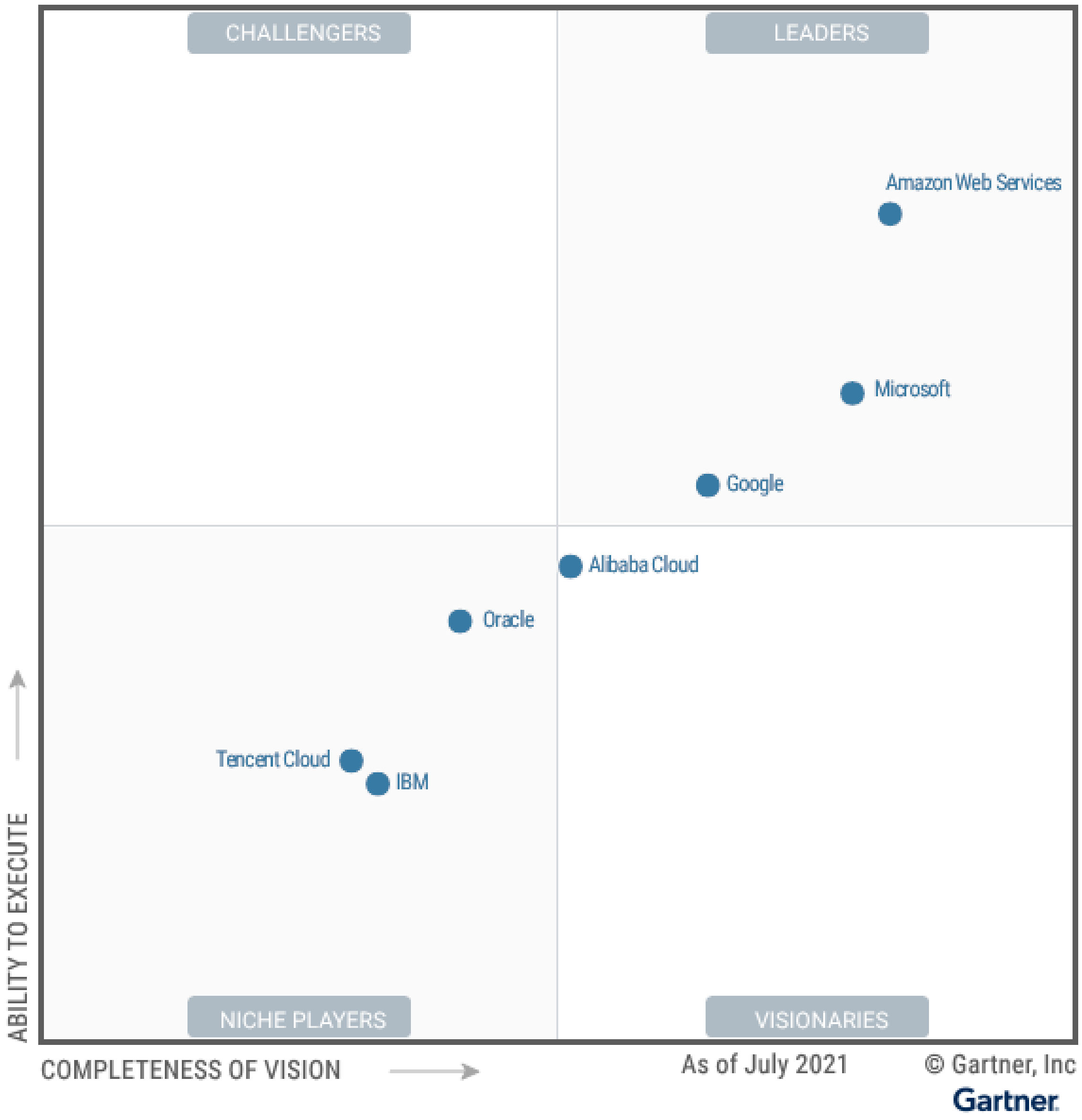


Figure 3: Magic Quadrant for Cloud Infrastructure and Platform Services

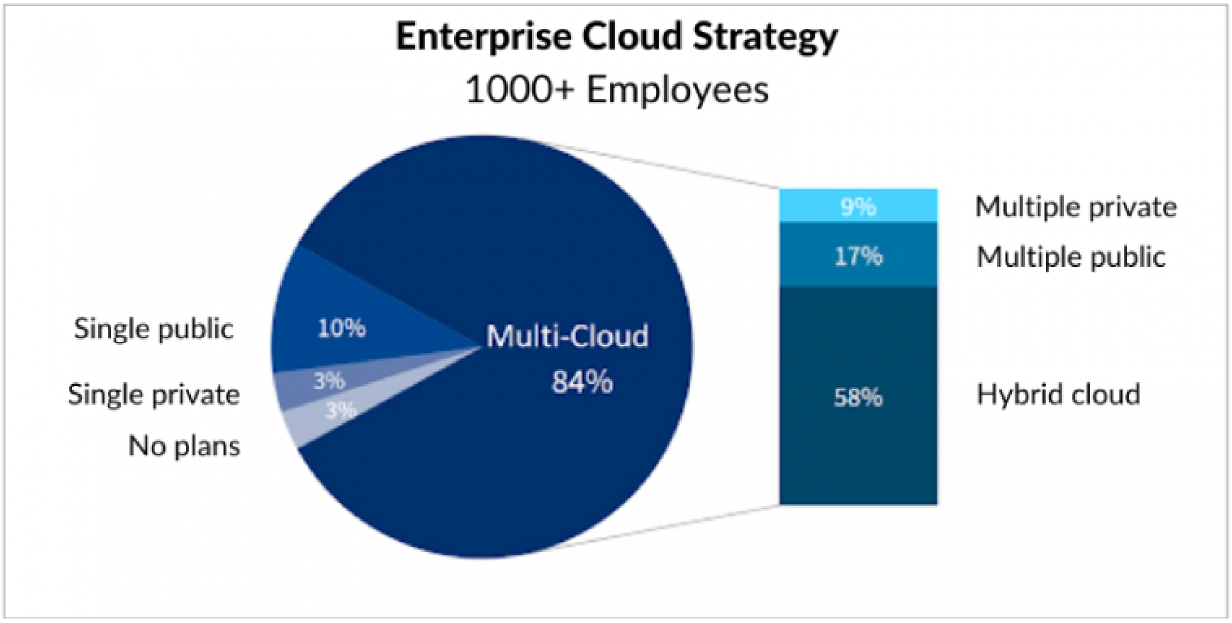


Figure 4: Cloud Strategy - image source Flexera

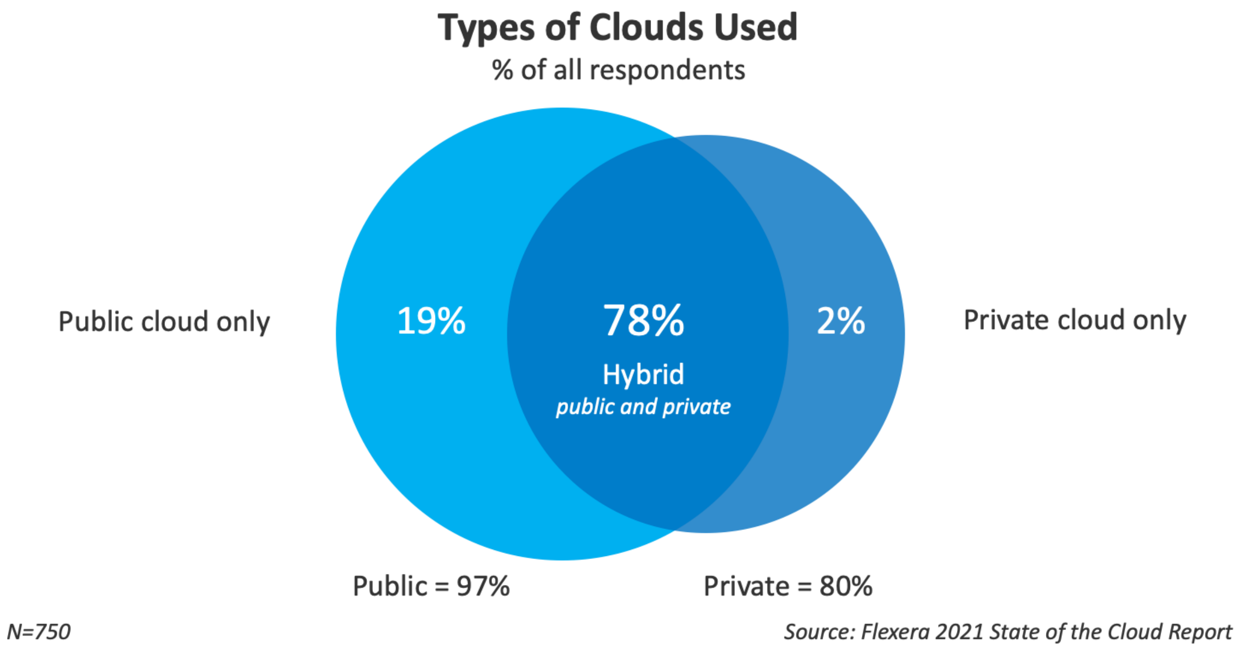
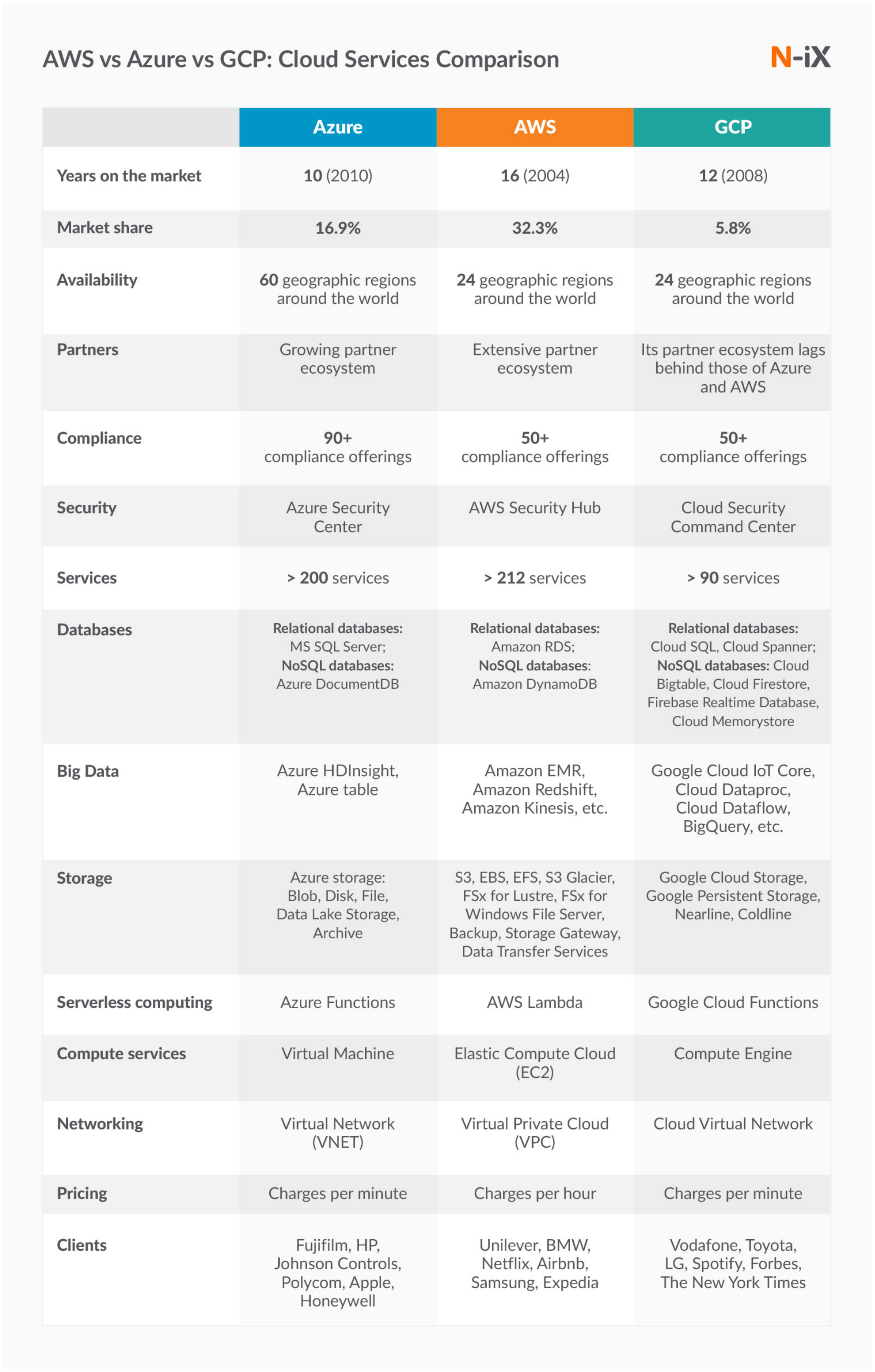


Figure 5: Cloud Strategy - image source Flexera

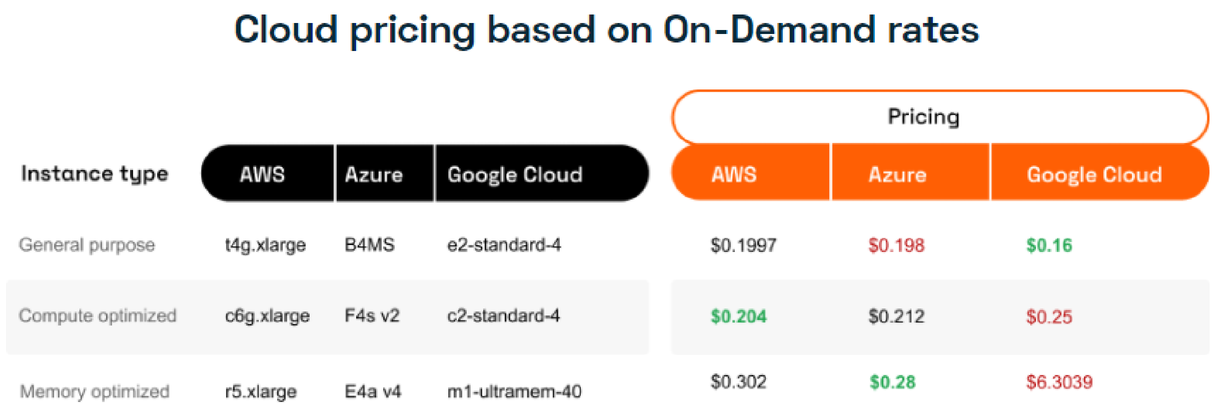


AWS vs Azure vs GCP: Cloud Services Comparison

N-iX

	Azure	AWS	GCP
Years on the market	**10** (2010)	**16** (2004)	**12** (2008)
Market share	**16.9%**	**32.3%**	**5.8%**
Availability	**60** geographic regions around the world	**24** geographic regions around the world	**24** geographic regions around the world
Partners	Growing partner ecosystem	Extensive partner ecosystem	Its partner ecosystem lags behind those of Azure and AWS
Compliance	**90+** compliance offerings	**50+** compliance offerings	**50+** compliance offerings
Security	Azure Security Center	AWS Security Hub	Cloud Security Command Center
Services	**> 200** services	**> 212** services	**> 90** services
Databases	**Relational databases:** MS SQL Server; **NoSQL databases:** Azure DocumentDB	**Relational databases:** Amazon RDS; **NoSQL databases:** Amazon DynamoDB	**Relational databases:** Cloud SQL, Cloud Spanner; **NoSQL databases:** Cloud Bigtable, Cloud Firestore, Firebase Realtime Database, Cloud Memorystore
Big Data	Azure HDInsight, Azure table	Amazon EMR, Amazon Redshift, Amazon Kinesis, etc.	Google Cloud IoT Core, Cloud Dataproc, Cloud Dataflow, BigQuery, etc.
Storage	Azure storage: Blob, Disk, File, Data Lake Storage, Archive	S3, EBS, EFS, S3 Glacier, FSx for Lustre, FSx for Windows File Server, Backup, Storage Gateway, Data Transfer Services	Google Cloud Storage, Google Persistent Storage, Nearline, Coldline
Serverless computing	Azure Functions	AWS Lambda	Google Cloud Functions
Compute services	Virtual Machine	Elastic Compute Cloud (EC2)	Compute Engine
Networking	Virtual Network (VNET)	Virtual Private Cloud (VPC)	Cloud Virtual Network
Pricing	Charges per minute	Charges per hour	Charges per minute
Clients	Fujifilm, HP, Johnson Controls, Polycom, Apple, Honeywell	Unilever, BMW, Netflix, Airbnb, Samsung, Expedia	Vodafone, Toyota, LG, Spotify, Forbes, The New York Times

Figure 6: Cloud Services comparison - image source N-IX.com (2020)



Cloud pricing based on On-Demand rates

Instance type	AWS	Azure	Google Cloud	Pricing AWS	Pricing Azure	Pricing Google Cloud
General purpose	t4g.xlarge	B4MS	e2-standard-4	$0.1997	$0.198	$0.16
Compute optimized	c6g.xlarge	F4s v2	c2-standard-4	$0.204	$0.212	$0.25
Memory optimized	r5.xlarge	E4a v4	m1-ultramem-40	$0.302	$0.28	$6.3039

Figure 7: Pricing comparison - image source cast.ai

- while aiming for full automation could seem an overwhelming task, this will significantly reduce the time spent in the future and the challenge of managing these applications (?).

According to Forbes, there are now 77 % of organizations, having one or some parts of their systems in the cloud. The budget is usually allocated through multiple services. E.g. in 2018 on average, the distribution of the budget accordingly: 48% went to SaaS, 30% to IaaS, and 21% to PaaS.

1.3 Kubernetes in a nutshell

While virtualized applications are highly preferred as opposed to IaaS approach (virtual machines), it makes sense to dig deeper in kubernetes and docker setup, regardless of the chosen managed service.

Kubernetes was founded by Ville Aikas, Joe Beda, Brendan Burns, and Craig McLuckie in collaboration with Google engineers Brian Grant and Tim Hockin in mid-2014. Google's Borg system heavily influenced kubernetes design (?) (?). While the Borg project was implemented entirely in C++, Kubernetes was rewritten in Go language. The main goal of kubernetes was to build on the capabilities of containers and provide significant gains in programmer productivity while easing the management of the system.

Kubernetes is the most popular container orchestration platform that enables users to create and run multiple containers in cloud environments. Kubernetes offers resource management to isolate the resource usage of containers on a host server because performance isolation is an important factor in terms of service quality.

2 Public Cloud Setup

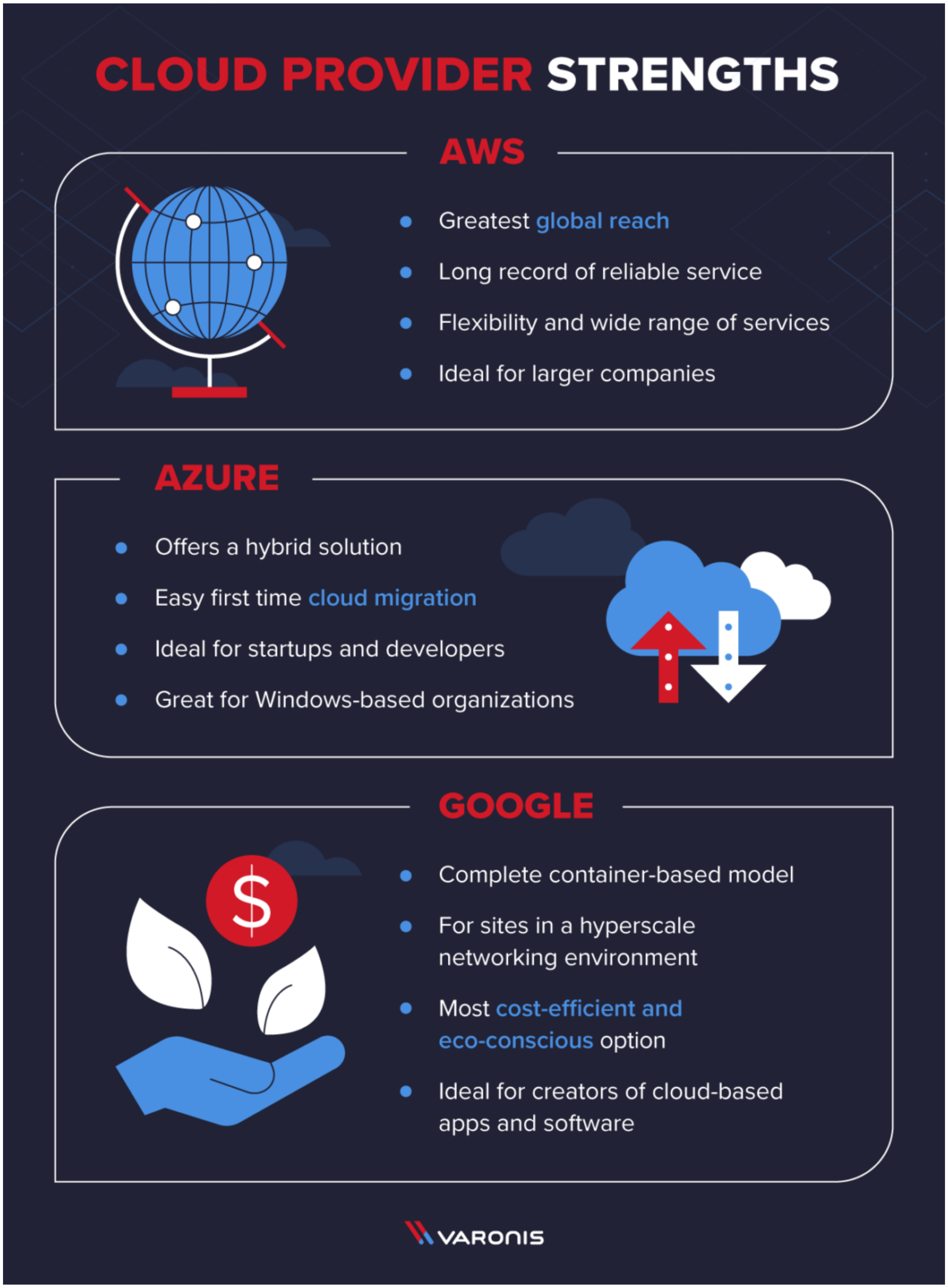
2.1 Security

Cloud security is a critical matter. Most companies worry that highly sensitive data and intellectual property may be exposed through accidental leaks or due to increasingly sophisticated cyber attacks. Gartner predicts that through 2025, 99% of cloud security failures will be the customer's fault.

Moreover, having a solid cloud security stance helps organizations achieve other benefits, such as:

- Lower costs
- Reduced ongoing operational and administrative expenses
- Scalability
- Increased reliability and availability
- DevOps way of working

Despite bringing many benefits, the cloud computing paradigm imposes serious concerns in terms of security



CLOUD PROVIDER STRENGTHS

AWS

- Greatest global reach
- Long record of reliable service
- Flexibility and wide range of services
- Ideal for larger companies

AZURE

- Offers a hybrid solution
- Easy first time cloud migration
- Ideal for startups and developers
- Great for Windows-based organizations

GOOGLE

- Complete container-based model
- For sites in a hyperscale networking environment
- Most cost-efficient and eco-conscious option
- Ideal for creators of cloud-based apps and software

VARONIS

Figure 8: Conclusion - image source www.varonis.com

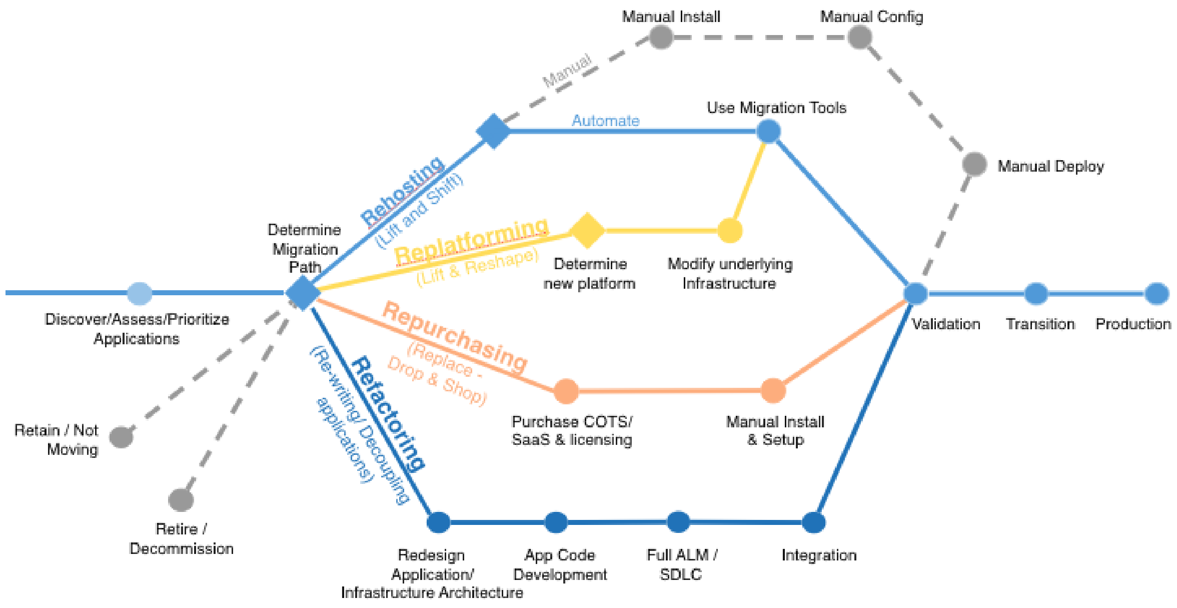


Figure 9: Cloud migration strategy - image source aws.amazon.com

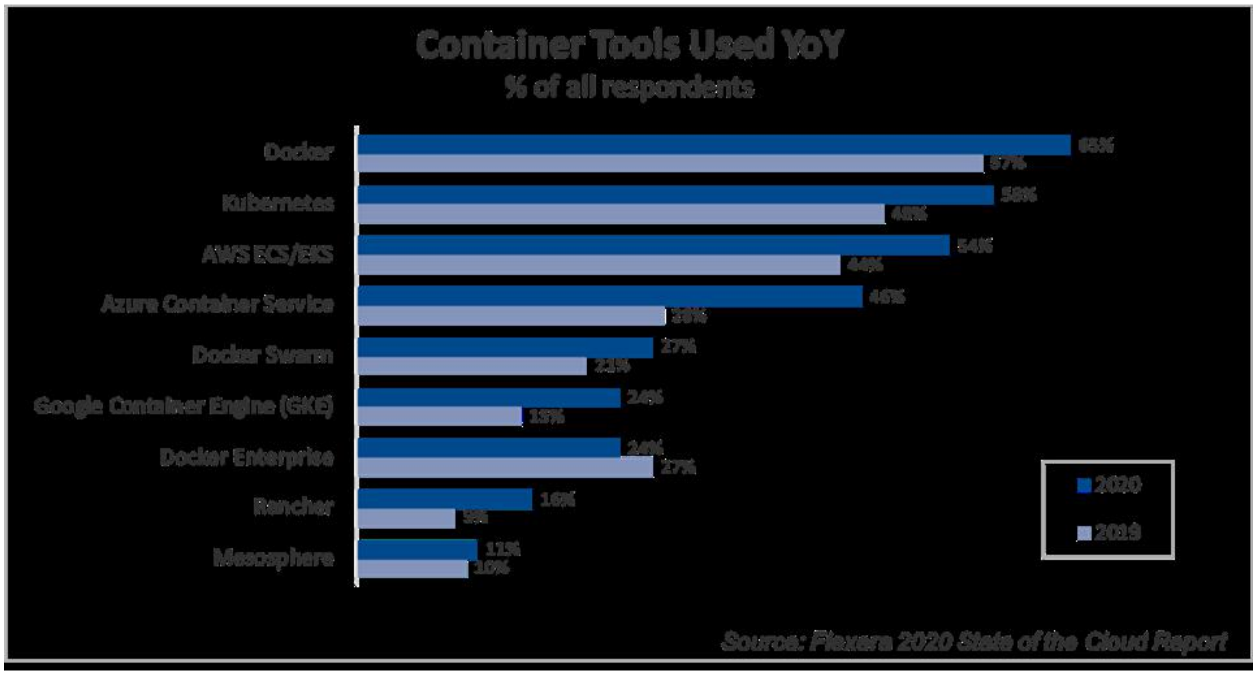


Figure 10: Cloud Strategy - image source Flexera

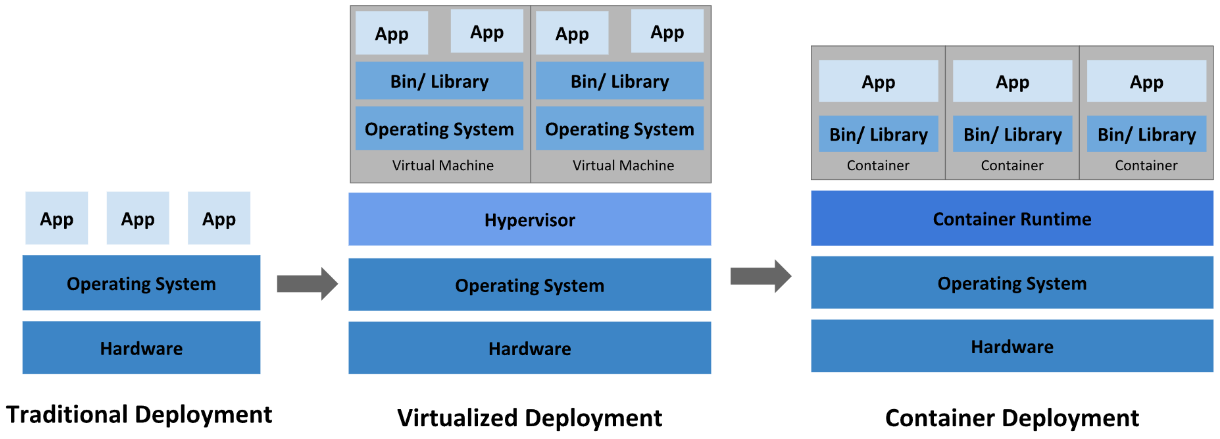


Figure 11: Container evolution - kubernetes.io

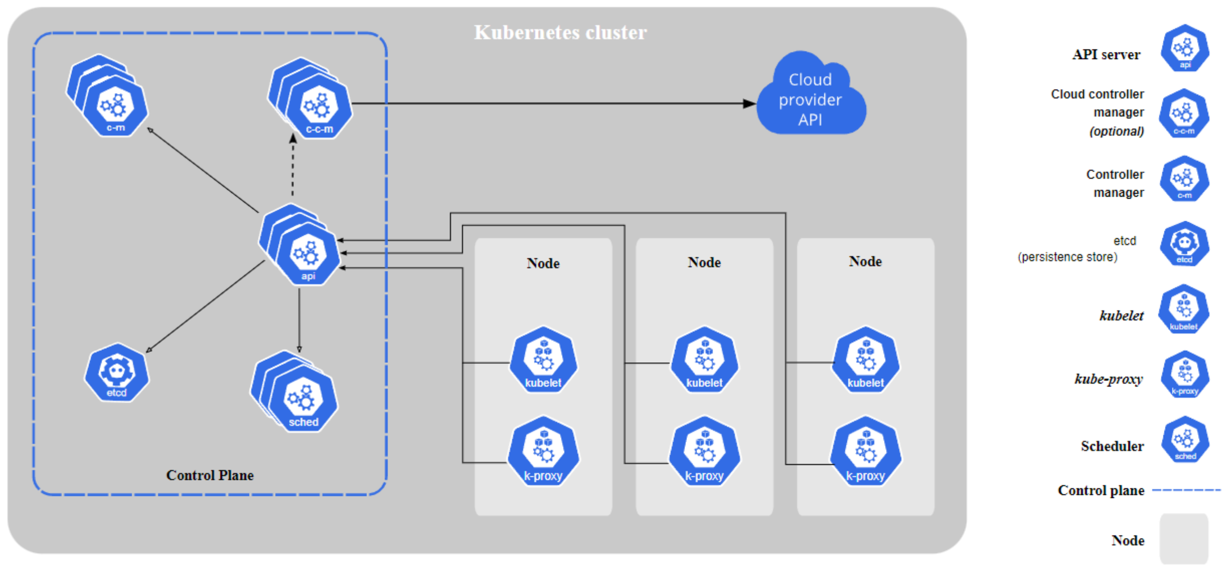


Figure 12: The components of a Kubernetes cluster - kubernetes.io

and privacy, which are considered hurdles in the adoption of the cloud at a very large scale (?). Security issues are depended on the cloud provider, service user, instance (?) and the delivery model, PaaS, IaaS, and SaaS (?). Data stored in public cloud would face both outside attacks and inside attacks (?). Data loss and leakage were as the biggest security concern, with 44% of organizations seeing data loss as one of their top three focus areas. Two-thirds of organizations leave back doors open to attackers leading to an accidental exposure through misconfiguration. Security gaps in misconfigurations were exploited in 66% of attacks (?).

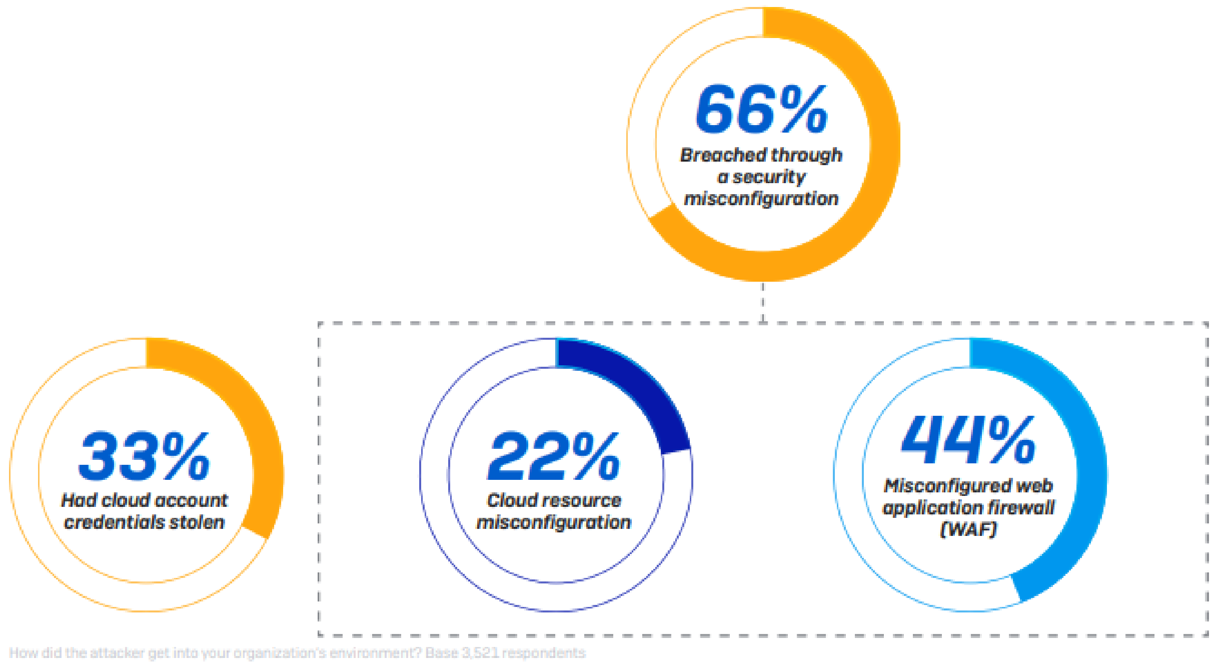


Figure 13: How criminals are getting in, source - sophos.com

Zero Trust security model enables securing cloud native applications by encrypting all network communication, authenticating, and authorizing every request. The traditional trust management mechanisms represent a static trust relationship which falls deficit while meeting up the dynamic requirement of cloud services. (?). In order to to achieve a true zero-trust security model in the cloud, a combination of network and identity permission policies should be in place.

To adequately address the modern dynamic threat environment requires(?):

- Coordinated and aggressive system monitoring, system management, and defensive operations capabilities.
- Assuming all requests for critical resources and all network traffic may be malicious.
- Assuming all devices and infrastructure may be compromised.
- Accepting that all access approvals to critical resources incur risk

Some security recommendations for network security can be summarized as follows (?): * Secure communication techniques should be adopted: HTTPS for web applications, transmission channel must be encrypted by TLS * Additional monitoring should be done (manual, automatic, ML based) * Other public security services such as web application firewalls (WAF), virtual firewalls, virtual bastion machines, virtual host protection and virtual database audit systems could be used

2.2 Infrastructure as Code

There was a significant shift in development, deployment and software application management during the past decade. The new approach is called Development Operations (DevOps) where Infrastructure as Code

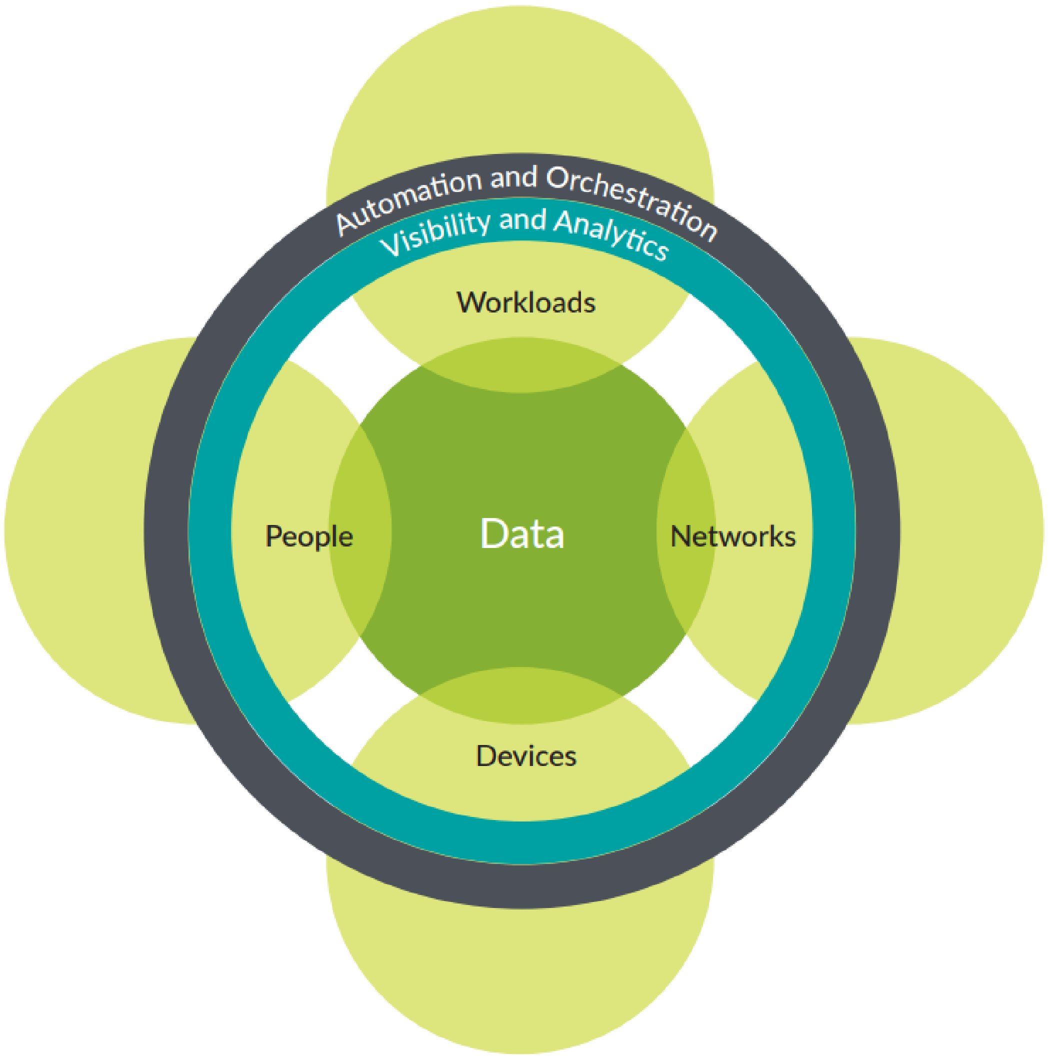


Figure 14: The Zero Trust eXtended (ZTX) Ecosystem, Forrester Research, Inc., source - juniper.net

(IaC) plays a core role. While manual configurations in Cloud context was a norm, nowadays it is fully automated using blueprints that are easily interpretable by machines. Moreover, IaC approach allows a faster and homogeneous configuration for the whole infrastructure. Usually, it is utilized by a specific declarative languages (TerraForm, CloudFormation, Puppet) that allows users to describe the desired state of the infrastructure. This significantly reduces the time, complexity and helps to provision the infrastructure from the security, management and costs perspectives.

The whole idea behind IaC is simple - developers can write declarative statements that define the infrastructure necessary to run the code as oppose to writing a ticket/creating a task for administrators. Reproducibility and transparency comes as a side effect.

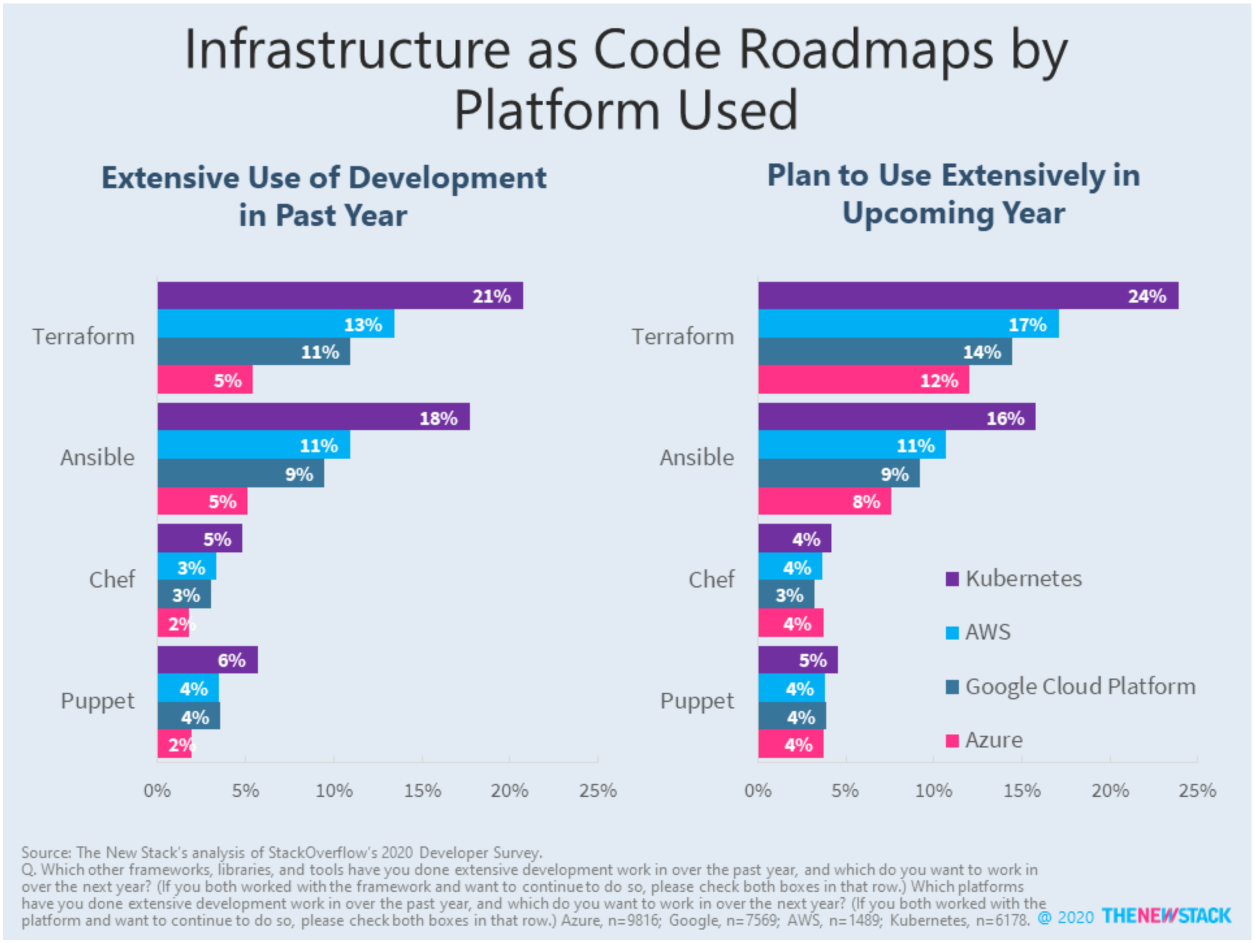


Figure 15: Infrastructure as Code Survey, source - thenewstack.io

Terraform is one of the most popular way to implement this pipeline, especially in a Cloud context. It is an is an open source tool that lets you provision Google Cloud resources with declarative configuration files—resources such as virtual machines, containers, storage, and networking. It lets uers manage Terraform configuration files in source control to maintain an ideal provisioning state for testing, production and other environments. (?)

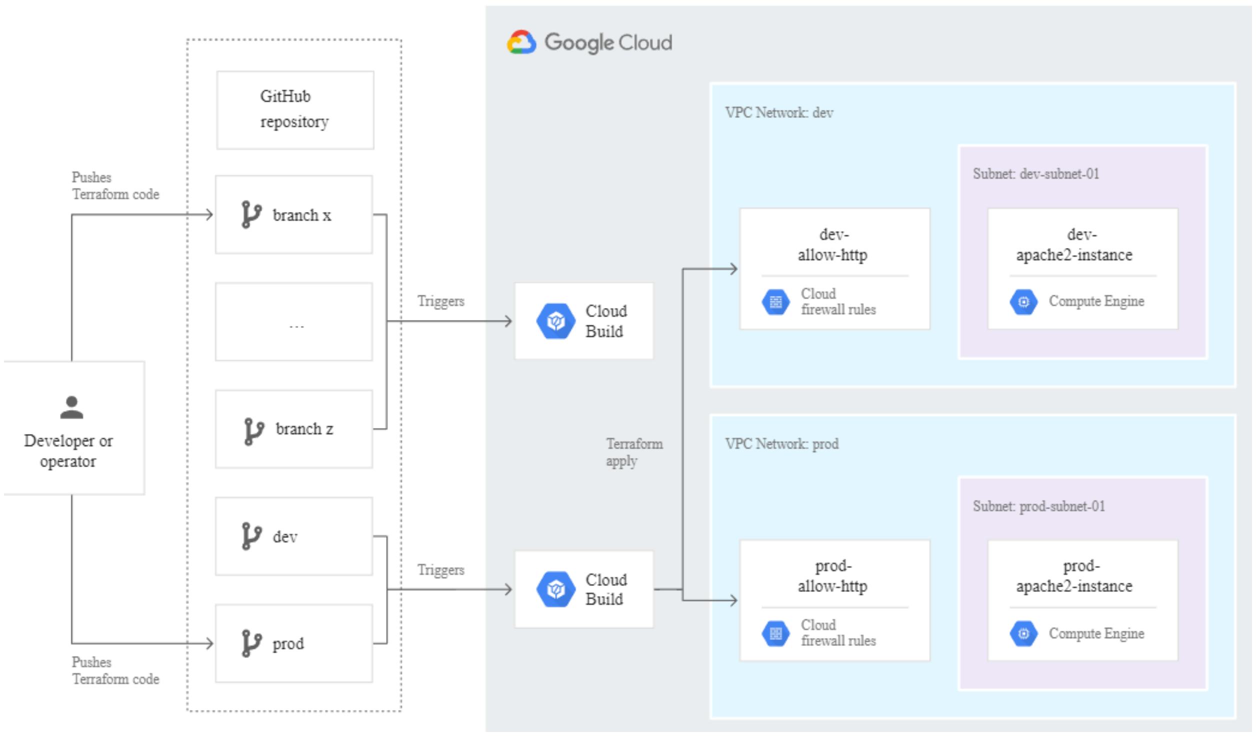


Figure 16: Terraform example, source - cloud.google.com

3 Use Cases

3.1 Shiny Server on Cloud Run

R is a programming language and free software environment for statistical computing and graphics. It is widely used among statisticians and data miners for developing statistical software and data analysis. R is usually used internally, mostly for interactive analysis and statistical modeling, but recently there are more and more applications in terms of WEB applications and APIs. While R is not the most popular language, it has no luxury of possessing of the box serving platform in most of the cloud providers. However, it can be nicely integrated with docker having all the dependencies in place and encapsulated application in a single container. This practice greatly speeds up the workflow of software development and deployment. In this proof of concept we will suggest the best approach of migrating R applications, shiny apps, and APIs having a cloud provider selected (GCP).

Google Kubernetes Engine (GKE) is a great choice for a container orchestration platform and offers advanced scalability and configuration flexibility. GKE gives a complete control over every aspect of container orchestration, from networking to storage, to how you set up observability in addition to supporting stateful application use cases. A fully managed Cloud Run is the additional service based on GKE for those applications which do not need a comprehensive level of cluster configuration and monitoring. Additionally, the serverless approach provides more fine-grained billing and can significantly reduce the cost (e.g. in case the application is not in use). With a manually created GKE cluster, the nodes and environment are always on which means that you are billed for them regardless of utilization. With Cloud Run, the service is merely available and the billing is done only for the actual consumption.

There are even more reasons to choose Cloud Run instead of Kubernetes cluster. Typically R users are not software engineers, so we should not only aim for simplicity in the development flow, but convenient application management as well (e.g. Google Cloud Run application automatically scales up depending on the traffic). Cloud Run is also integrated with Stackdriver Monitoring, Logging, and Error Reporting services.

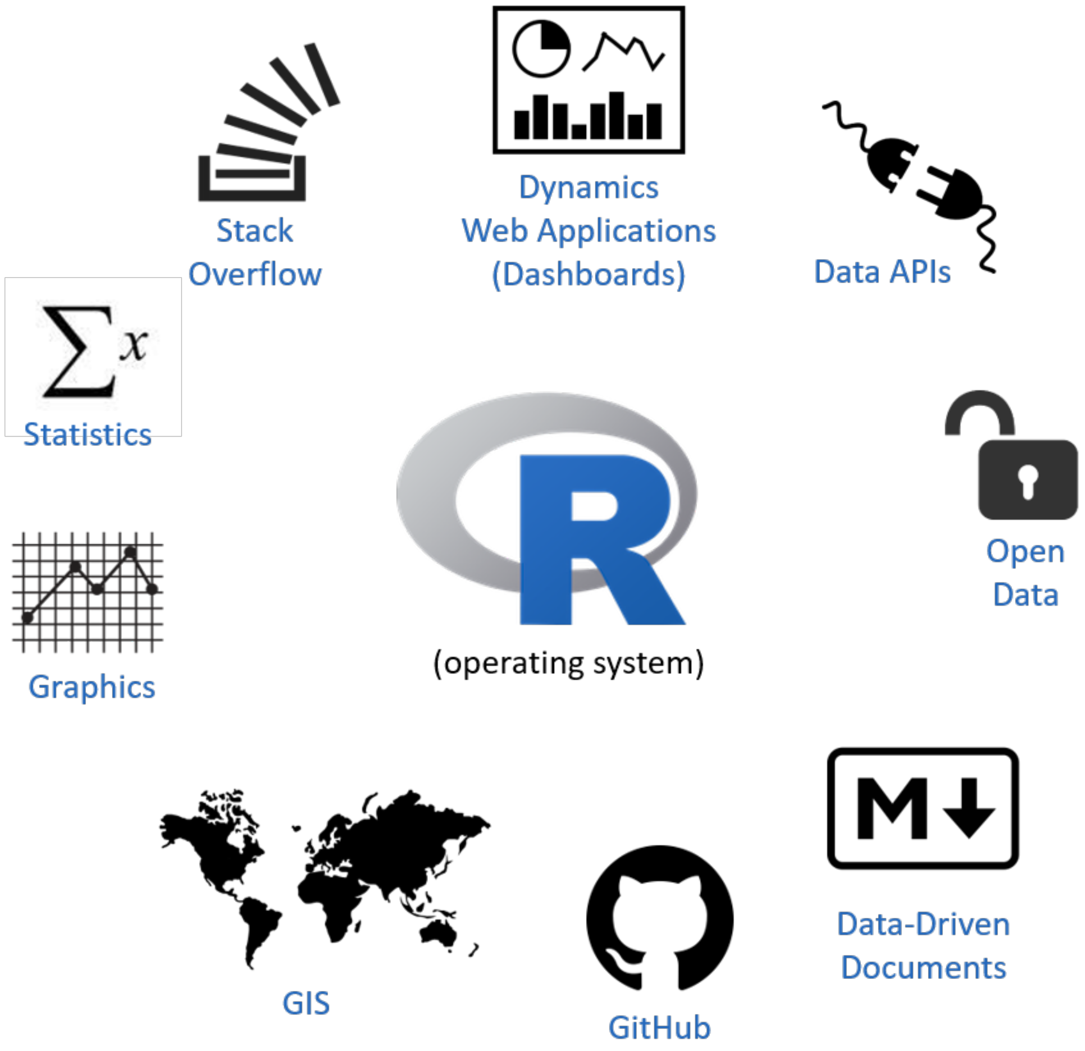


Figure 17: R applications - image source ds4ps.org

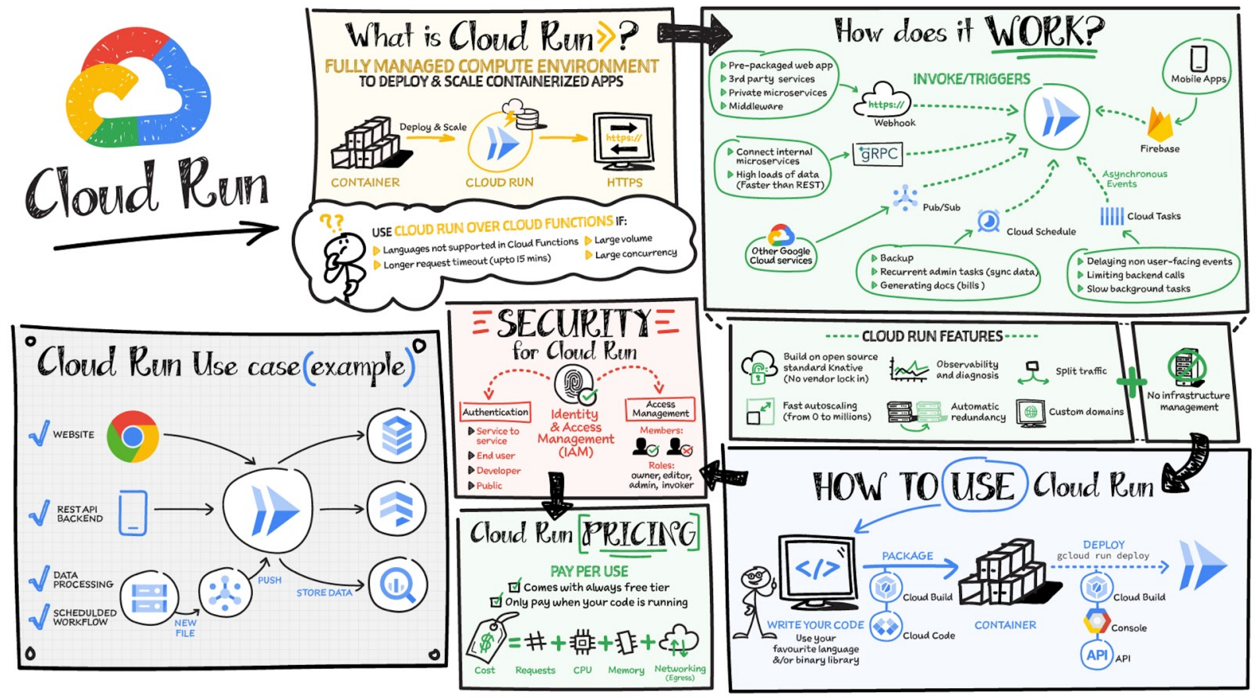


Figure 18: Cloud Run - cloud.google.com

Moreover, Cloud Run is is constructed on the Knative open-source project, thus enabling the portability of the workflows.

Let's take the Shiny app example. First we need to build an app and containerize it. Since Google Cloud Run was introduced only in 2019, there were only a few attempts to leverage this technology against Shiny R applications (?) (?). Once application is finished, Dokerfile is needed with all the dependencies included:

```
FROM rocker/shiny-verse:latest
RUN apt-get update && apt-get install -y \
    sudo \
    pandoc \
    pandoc-citeproc \
    libcurl4-gnutls-dev \
    libcairo2-dev \
    libxt-dev \
    libssl-dev \
    libssh2-1-dev
RUN R -e "install.packages('shinydashboard', repos='http://cran.rstudio.com/')"
COPY shiny-server.conf  /etc/shiny-server/shiny-server.conf
COPY /app /srv/shiny-server/
RUN rm /srv/shiny-server/index.html
EXPOSE 80
COPY shiny-server.sh /usr/bin/shiny-server.sh
RUN ["chmod", "+x", "/usr/bin/shiny-server.sh"]
CMD ["/usr/bin/shiny-server.sh"]
```

Once the docker image is built, all we need to do is to push that image to Google Container Registry.

From there, we can deploy the docker image manually or use other tools such as Terraform to create infrastructure as code. Since deployment on Google Cloud Run is quite simple, Terraform script is not complex.

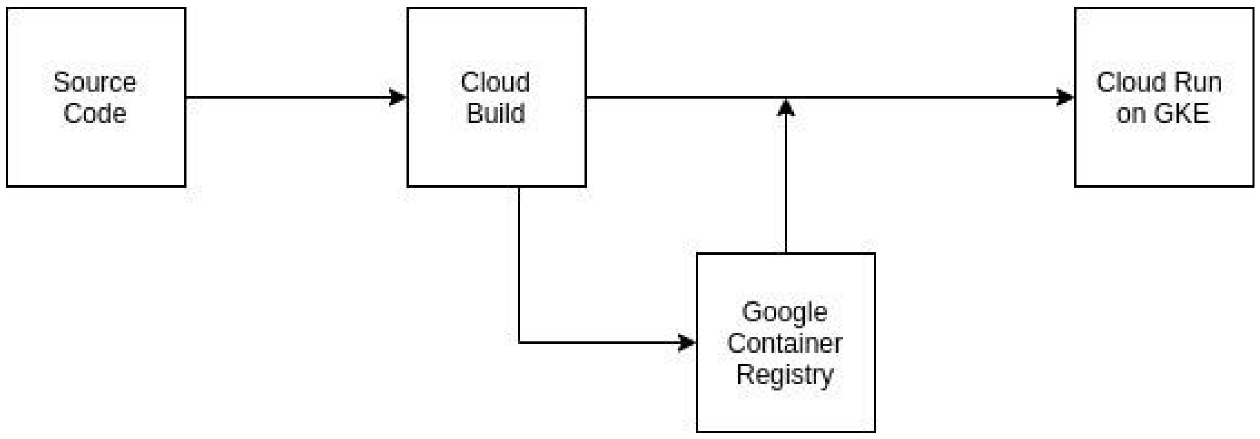


Figure 19: Cloud Run Process - image source https://medium.com/google-cloud

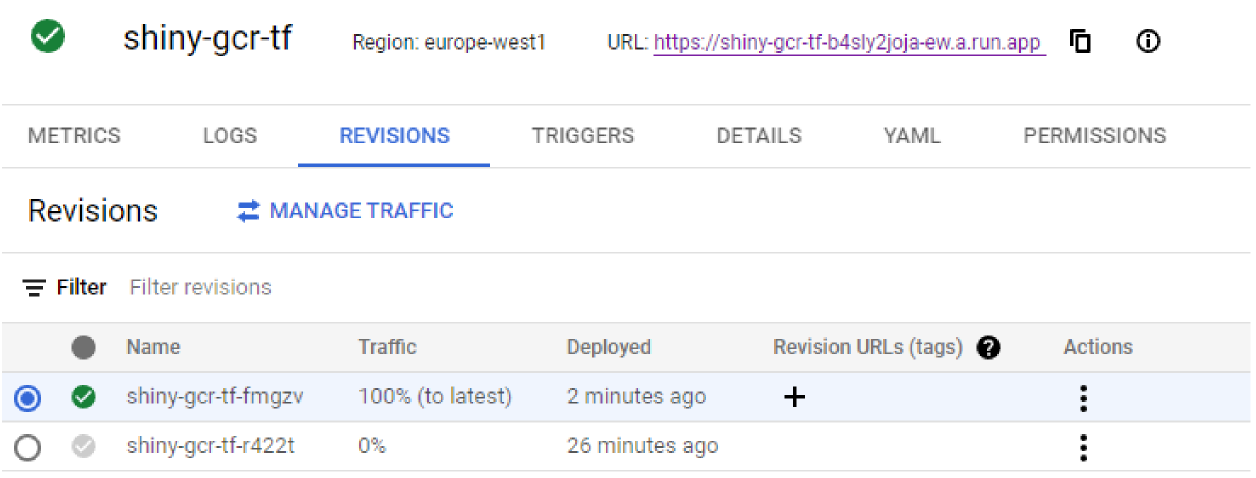
Terraform script source - github.

```
provider "google"{
 project     = "vgtu-cloud"
 region      = "europe-west1"
}

resource "google_project_service" "run" {
  service = "run.googleapis.com"
}

resource "google_cloud_run_service" "shiny-gcr-tf" {
  name = "shiny-gcr-tf"
  location = "europe-west1"
  template {
    spec {
      containers {
        image = "gcr.io/vgtu-cloud/mydashboard@sha256:3b1d046e01694a681664b350c62fa0b55e69bdd8e4f068642
        resources {
          limits = {
            cpu = "1000m"
            memory = "1024Mi"
          }
        }
        ports {
          container_port = 80
        }
      }

      container_concurrency = 80
      timeout_seconds = 300
    }
    metadata {
      annotations = {
        "autoscaling.knative.dev/minScale" = 0
        "autoscaling.knative.dev/maxScale" = 1
      }
    }
  }

  traffic {
    percent = 100
    latest_revision = true
  }

  depends_on = [google_project_service.run]
}

resource "google_cloud_run_service_iam_member" "allUsers" {
  service  = google_cloud_run_service.shiny-gcr-tf.name
  location = google_cloud_run_service.shiny-gcr-tf.location
  role     = "roles/run.invoker"
  member   = "allUsers"
}
```

Terraform script is initiated by the command line, but could be later reused in any CI/CD pipeline.

```
terraform init
terraform plan
terraform apply
```

It is easy to monitor and track deployments in Google Cloud Console interface. Metrics are tracked separately as well.



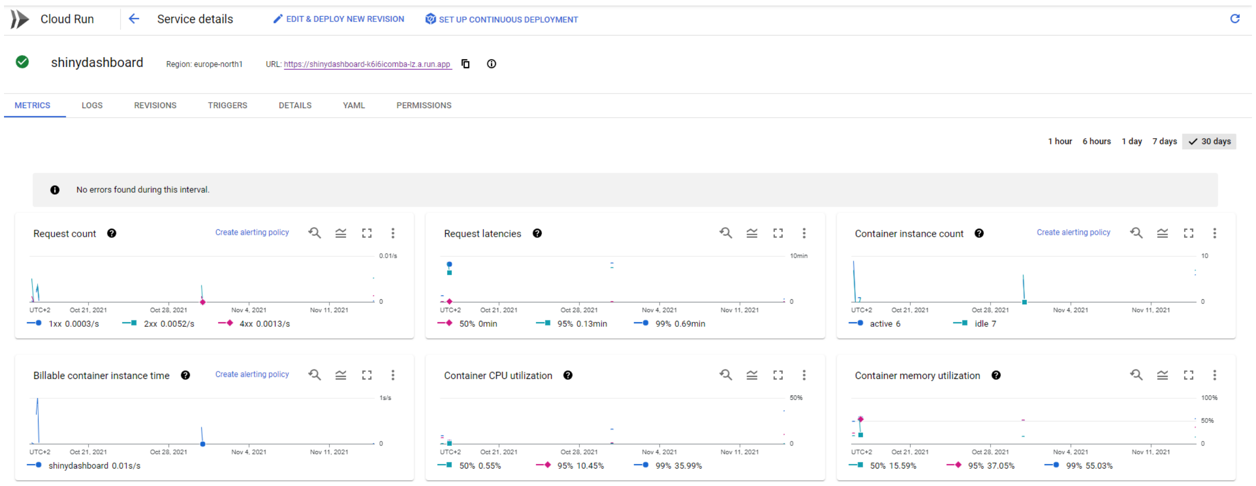
		Name	Traffic	Deployed	Revision URLs (tags)	Actions
●	✓	shiny-gcr-tf-fmgzv	100% (to latest)	2 minutes ago	+	⋮
○	✓	shiny-gcr-tf-r422t	0%	26 minutes ago		⋮

Figure 20: Cloud Run Deployments

It is important to note that Cloud Run is the cheapest and the most convenient option for Shiny R Application deployment. Minimum instances could be set to 0 to make this application offline when it is not in use. Moreover, SSL certificates are applied automatically and are handled by GCP. Application could be reached publicly using this URL: https://shiny-gcr-tf-b4sly2joja-ew.a.run.app/

3.2 Shiny Server on GKE

Google Kubernetes Engine is a straightforward way of setting up a Kubernetes Cluster. These clusters are fully managed by Google Site Reliability Engineers and Google ensures that your cluster is available and up-to-date. It also supports the common Docker container format and runs on Container-Optimized OS.



No errors found during this interval.

Request count: 1xx 0.0003/s; 2xx 0.0052/s; 4xx 0.0013/s

Request latencies: 50% 0min; 95% 0.13min; 99% 0.69min

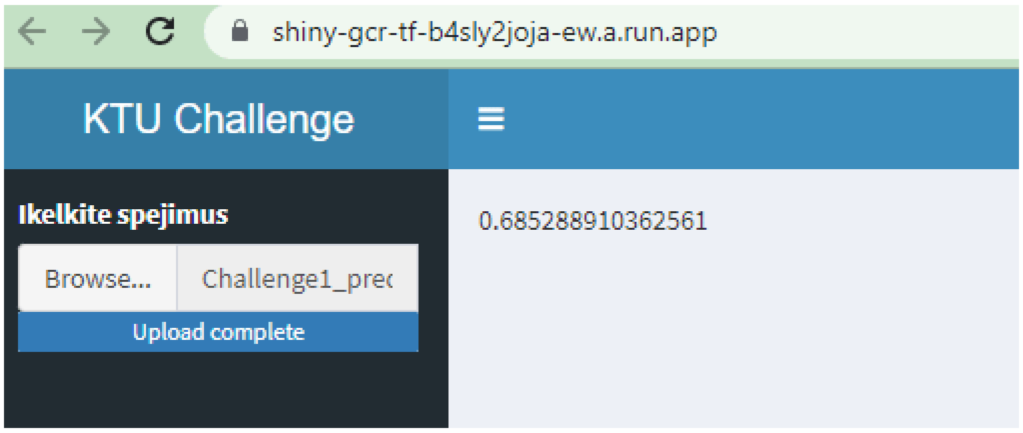
Container instance count: active 6; idle 7

Billable container instance time: shinydashboard 0.01s/s

Container CPU utilization: 50% 0.55%; 95% 10.45%; 99% 35.99%

Container memory utilization: 50% 15.59%; 95% 37.05%; 99% 55.03%

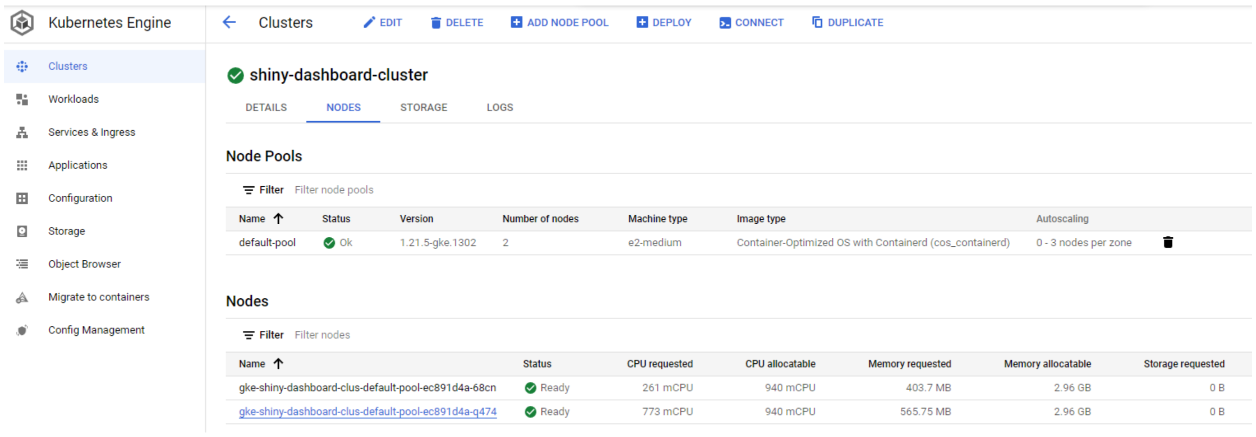
Figure 21: Cloud Run Metrics



0.685288910362561

Figure 22: Cloud Run Metrics

Once the Cluster is created, the deployment is similar to Cloud Run. It can be done via Google Cloud Console, GCP SDK or integrated in any CI/CD pipeline using Jenkins, GitActions or similar tools.



shiny-dashboard-cluster

Node Pools

Name	Status	Version	Number of nodes	Machine type	Image type	Autoscaling
default-pool	Ok	1.21.5-gke.1302	2	e2-medium	Container-Optimized OS with Containerd (cos_containerd)	0 - 3 nodes per zone

Nodes

Name	Status	CPU requested	CPU allocatable	Memory requested	Memory allocatable	Storage requested
gke-shiny-dashboard-clus-default-pool-ec891d4a-68cn	Ready	261 mCPU	940 mCPU	403.7 MB	2.96 GB	0 B
gke-shiny-dashboard-clus-default-pool-ec891d4a-q474	Ready	773 mCPU	940 mCPU	565.75 MB	2.96 GB	0 B

Figure 23: Shiny on GKE

This is an example of the same application deployed on GKE.

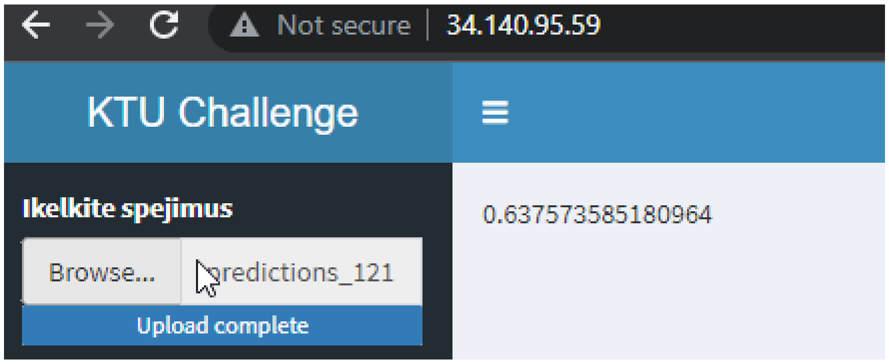


Figure 24: Shiny app in action

Load balancer - single point of entry

Rolling updates

Limitations

- Since Shiny is stateful application

3.3 Shiny Server on GCE

Google Compute Instance was selected as the third example and perfectly encapsulates Insfrastructure as a Service use case. There we need to prepare everything from scratch - use a fresh VM image of OS we want to use, install all the dependencies and applications we want to use. Later this virtual machine could serve as a centralized place to serve multiple Shiny R Applications.

The main.tf consists of region selection, virtual machine configuration (resources, OS, installed packages) and network (firewall) configuration:

```
provider "google" {
 project     = "vgtu-cloud"
 region      = "europe-west1"
```

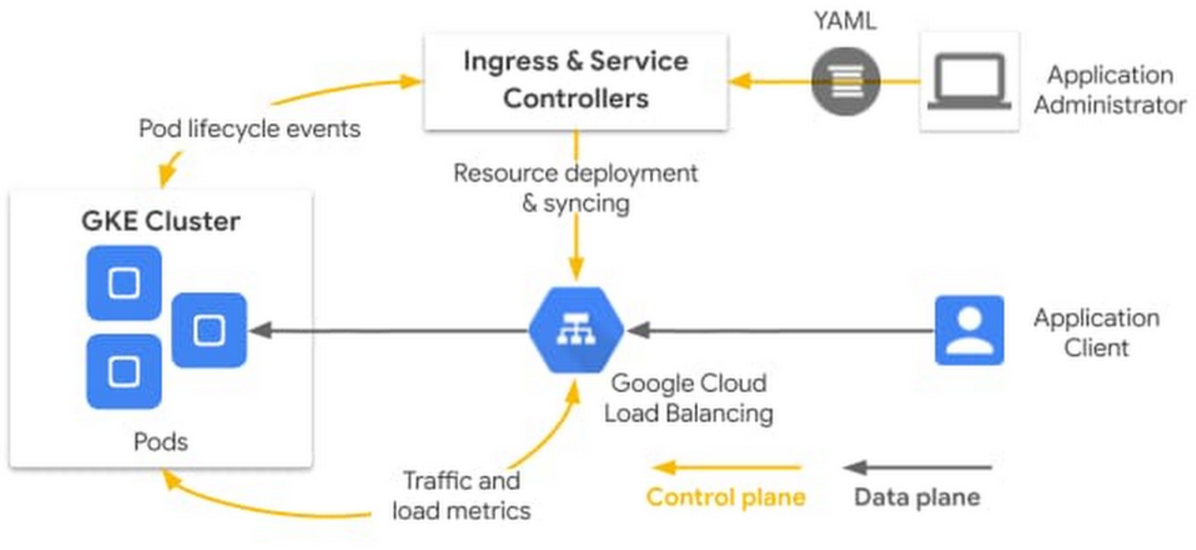

YAML
Ingress & Service Controllers
Application Administrator
Pod lifecycle events
Resource deployment & syncing
GKE Cluster
Pods
Google Cloud Load Balancing
Application Client
Traffic and load metrics
Control plane
Data plane

Figure 25: GCP Load Balancer

```
}

// Terraform plugin for creating random ids
resource "random_id" "instance_id" {
 byte_length = 8
}

// Google compute instance, Ubuntu Server - Europe region
resource "google_compute_instance" "default" {
 name         = "shiny-vm-${random_id.instance_id.hex}"
 machine_type = "f1-micro"
 zone         = "europe-west1-b"

 boot_disk {
   initialize_params {
     image = "ubuntu-2004-lts"
   }
 }

// Install R, Shiny Server open source and all the dependencies
 metadata_startup_script = "sudo apt-get update; sudo apt-get install -yq build-essential; sudo apt-get

 network_interface {
   network = "default"

   access_config {
     // Include this section to give the VM an external ip address
   }
 }
}
```

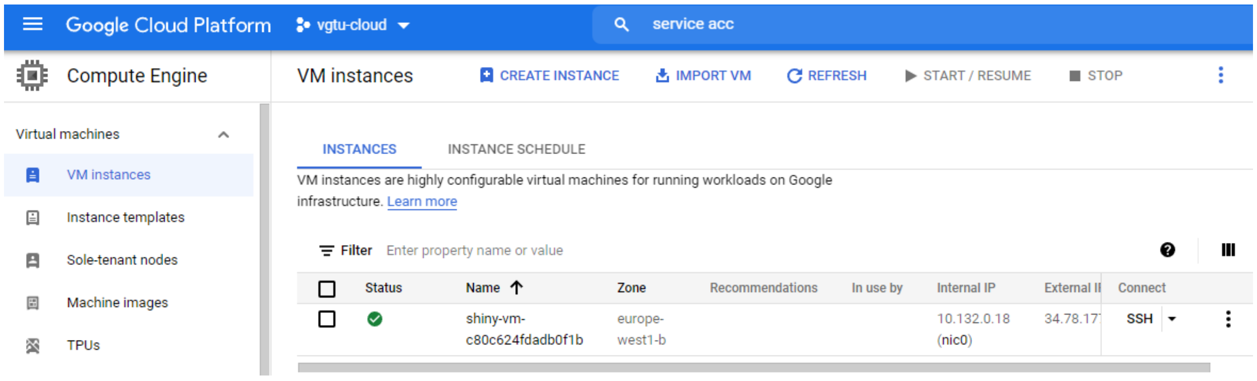
```
// Firewall exeptions
resource "google_compute_firewall" "default" {
 name    = "shiny-app-firewall"
 network = "default"
 source_ranges = ["0.0.0.0/0"]
 allow {
   protocol = "tcp"
   ports    = ["3838"]
 }
}
```

After the initiation, terraform generates a plan for the infrastructe. Once this plan is revised, it can be applied and resources will be created.

```
Terraform used the selected providers to generate the following execution
plan. Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # google_compute_firewall.default will be created
  + resource "google_compute_firewall" "default" {
      + creation_timestamp = (known after apply)
      + destination_ranges = (known after apply)
      + direction          = (known after apply)
      + enable_logging     = (known after apply)
      + id                 = (known after apply)
      + name               = "shiny-app-firewall"
      + network            = "default"
      + priority           = 1000
      + project            = (known after apply)
      + self_link          = (known after apply)
      + source_ranges      = [
          + "0.0.0.0/0",
        ]

      + allow {
          + ports    = [
              + "3838",
            ]
          + protocol = "tcp"
        }
    }

  # google_compute_instance.default will be created
  + resource "google_compute_instance" "default" {
      + can_ip_forward          = false
      + cpu_platform            = (known after apply)
      + current_status          = (known after apply)
      + deletion_protection     = false
      + guest_accelerator       = (known after apply)
      + id                      = (known after apply)
      + instance_id             = (known after apply)
      + label_fingerprint       = (known after apply)
      + machine_type            = "f1-micro"
      + metadata_fingerprint    = (known after apply)
      + metadata_startup_script = "sudo apt-get update; sudo apt-get install -yq build-essential python
      + min_cpu_platform        = (known after apply)
      + name                    = (known after apply)
      + project                 = (known after apply)
      + self_link               = (known after apply)
      + tags_fingerprint        = (known after apply)
      + zone                    = "europe-west1-b"

      + boot_disk {
          + auto_delete                = true
          + device_name                = (known after apply)
          + disk_encryption_key_sha256 = (known after apply)
          + kms_key_self_link          = (known after apply)
          + mode                       = "READ_WRITE"
          + source                     = (known after apply)

          + initialize_params {
              + image  = "ubuntu-2004-lts"
              + labels = (known after apply)
              + size   = (known after apply)
              + type   = (known after apply)
            }
        }

      + confidential_instance_config {
          + enable_confidential_compute = (known after apply)
        }

      + network_interface {
          + ipv6_access_type   = (known after apply)
          + name               = (known after apply)
          + network            = "default"
          + network_ip         = (known after apply)
          + stack_type         = (known after apply)
          + subnetwork         = (known after apply)
          + subnetwork_project = (known after apply)

          + access_config {
              + nat_ip       = (known after apply)
              + network_tier = (known after apply)
            }
        }

      + reservation_affinity {
          + type = (known after apply)

          + specific_reservation {
              + key    = (known after apply)
              + values = (known after apply)
            }
        }

      + scheduling {
```

```
          + automatic_restart   = (known after apply)
          + min_node_cpus       = (known after apply)
          + on_host_maintenance = (known after apply)
          + preemptible         = (known after apply)

          + node_affinities {
              + key      = (known after apply)
              + operator = (known after apply)
              + values   = (known after apply)
            }
        }
    }

  # random_id.instance_id will be created
  + resource "random_id" "instance_id" {
      + b64_std     = (known after apply)
      + b64_url     = (known after apply)
      + byte_length = 8
      + dec         = (known after apply)
      + hex         = (known after apply)
      + id          = (known after apply)
    }

Plan: 3 to add, 0 to change, 0 to destroy.
```

After the terraform apply, virtual machine is created in Google Compute Instance having all the libraries installed and network configuration applied.



VM instances are highly configurable virtual machines for running workloads on Google infrastructure. Learn more

	Status	Name ↑	Zone	Recommendations	In use by	Internal IP	External II	Connect
☐	✓	shiny-vm-c80c624fdadb0f1b	europe-west1-b			10.132.0.18 (nic0)	34.78.17	SSH

Figure 26: CE instance

Shiny server is reachable for everyone. More granular firewall rules could be applied for not exposing it to the whole internet. Additional authentication and https protocol could be applied if needed (since Shiny Server Open Source version does not support HTTPS by default, this could be handled via other tools, such as Apache Server, Nginx or Google Load Balancer service).

3.4 Limitations

- VM image templates with preinstalled software can be used for faster deployments/replication
- Integration with git flow and Github actions - branches, depended on the environments, terraform state saved in GCS buckets
- More granularity in terraform scripts - modules for infrastructure, network, variables, outputs, etc.
- VPC peering could be used as an alternative to public IPs - it allows internal IP address connectivity

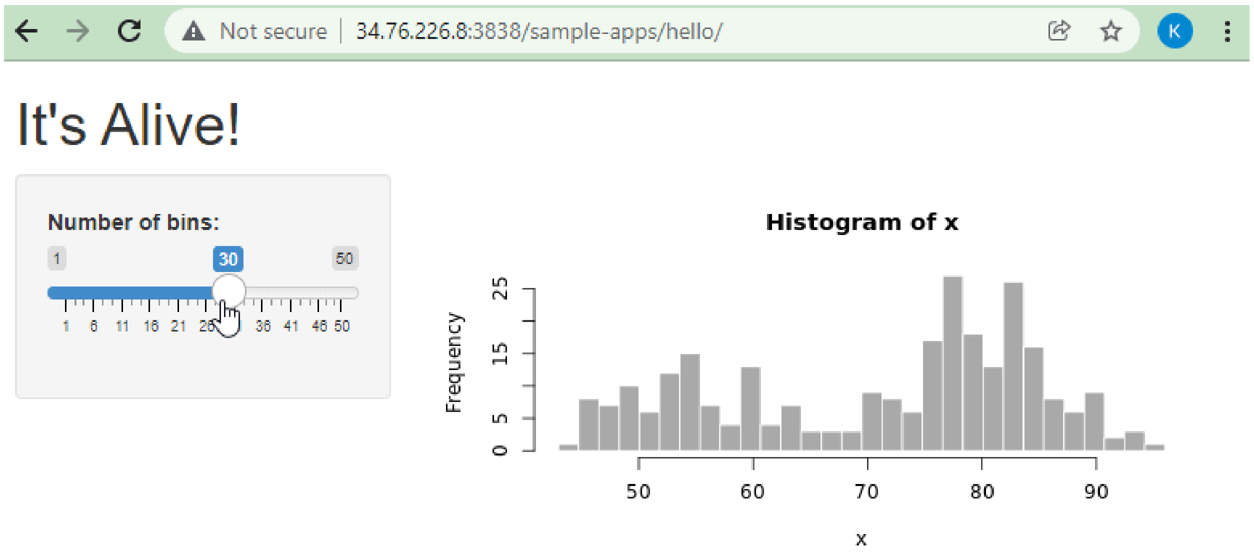


Figure 27: Shiny Server is Alive

across two Virtual Private Cloud (VPC) networks regardless of whether they belong to the same project or the same organization. Service owners do not need to have their services exposed to the public Internet and deal with its associated risks.

3.5 Security Experiment

Computer networks are prone to attacks and it has wide range of attacks associated with it. Cloud is not an exception and even holds more risk. It can be prone to Denial-of-service, Eavesdropping, Host Attacks, Password Guessing, Protocol-based and Social Engineering attacks (**?**).

As an experiment, firewall was opened to the whole world and network activity was monitored for one week. While the activity in the Compute Instance (Shiny Server hosted on a Virtual Server) was marginal, the exposed Shiny Server instance on Google Kubernetes Cluster was scanned extensively. This could be due to the rules how Google generate IP addresses for corresponsing instances. Moreover, GKE was exposed on port 80 which is a standard HTTP port, while standard port of shiny server (3838) was used for Compute Instance, which is not that common configuration.

While the majority of the requests came from USA, applications from China and Russia also scanned our exposed application considerably. These scans also are not centralized, but rather done by individuals or companies which specializes in data mining and web crawling. Some requests are also received from Lithuania, CGates Internet Service Provider.

Some of the IPs were crossed check with a publicly available IP database. These IP addresses, especially from China and Russia, were already reported a number of times and are indicated as abusive.

The analysis proves that incorrectly configured firewall poses one of the most significant security risk. Misconfigured applications could serve as a back door and is a low handing fruit for hackers - e.g. it is easy to run a port scan for a specific IP range and use a collection of scripts/exploits to check whether there are any holes in the application. If any sensitive data where General Data Protection Regulation is not applied (i.e. USA, China, Russia).

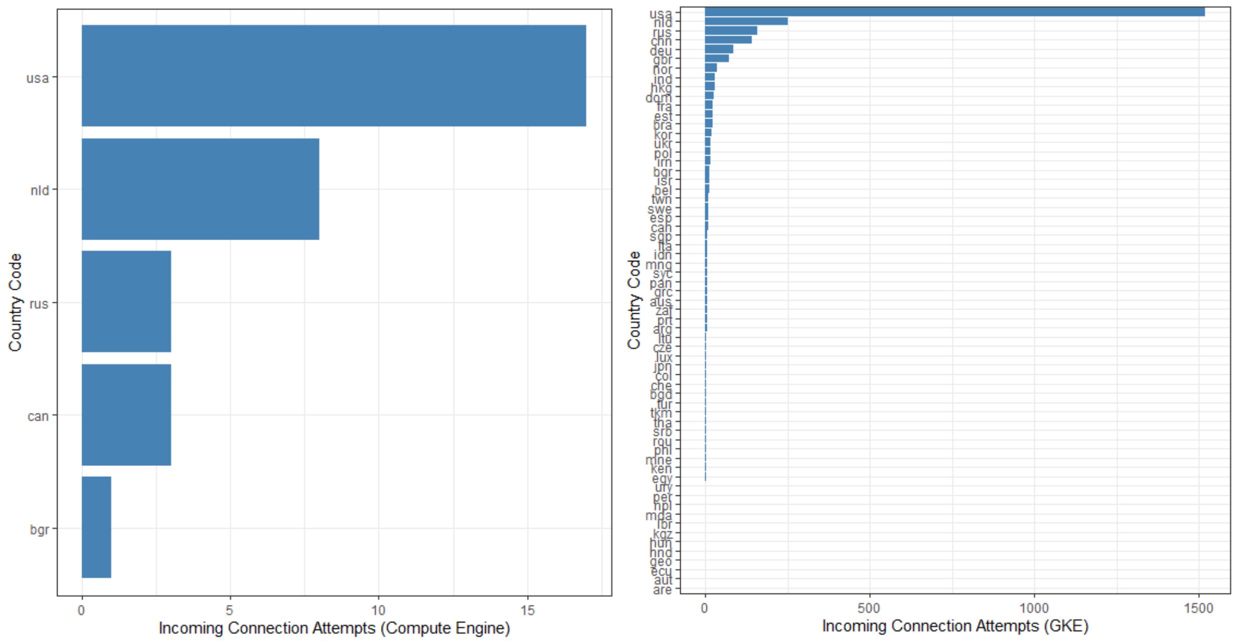


Figure 28: Incoming Requests

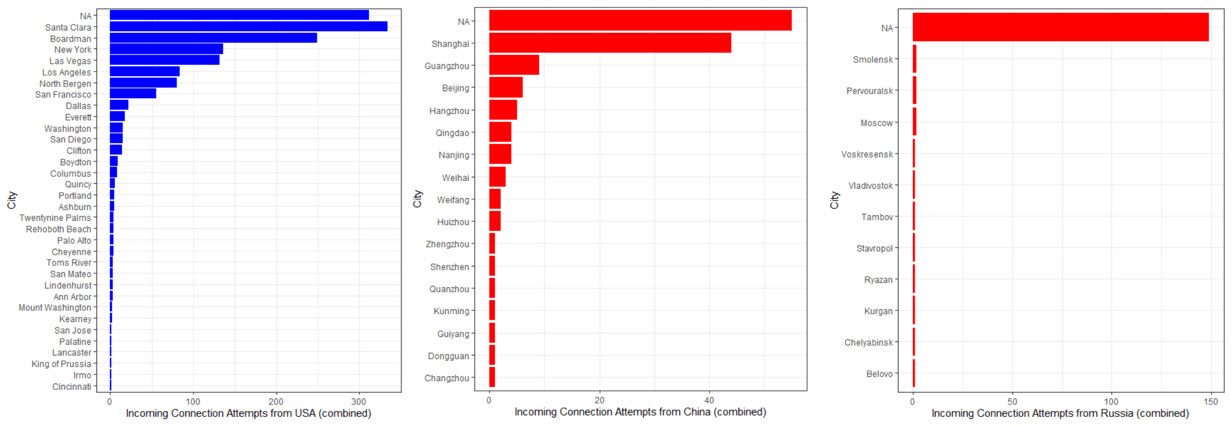


Figure 29: Incoming Requets from different cities