FMISD19004 Cloud Computing Technologies

Kęstutis Daugėla

2022-01-04

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 (**?**).

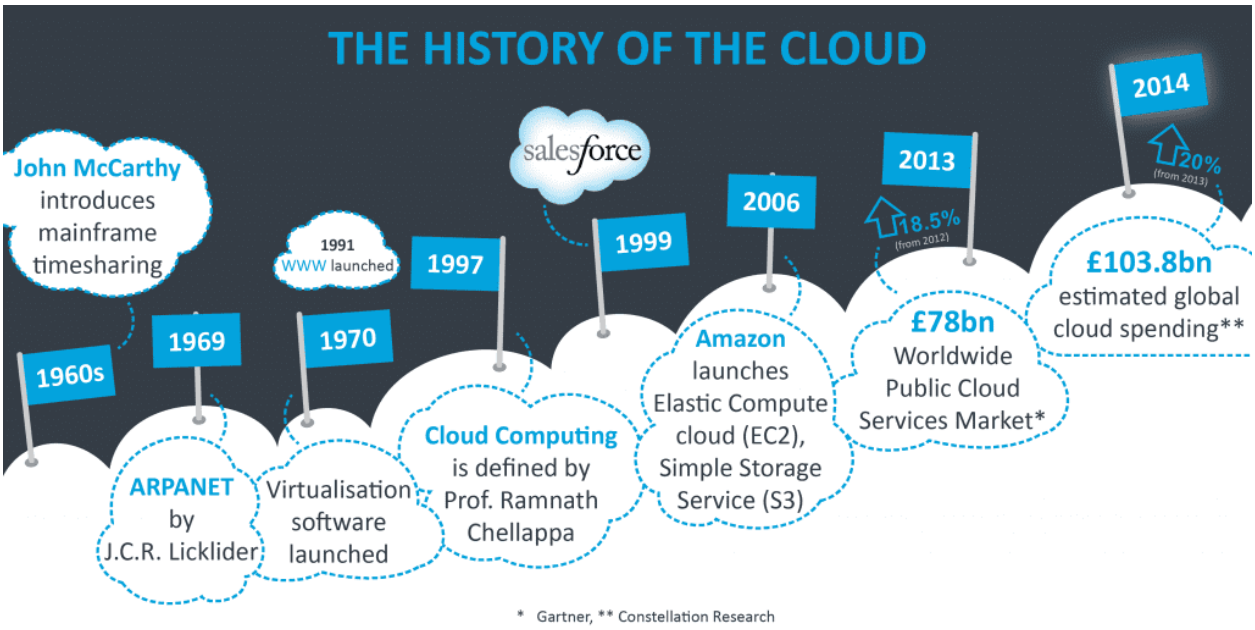

Figure 1: The history of the cloud - image source https://itchronicles.com/

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
- 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

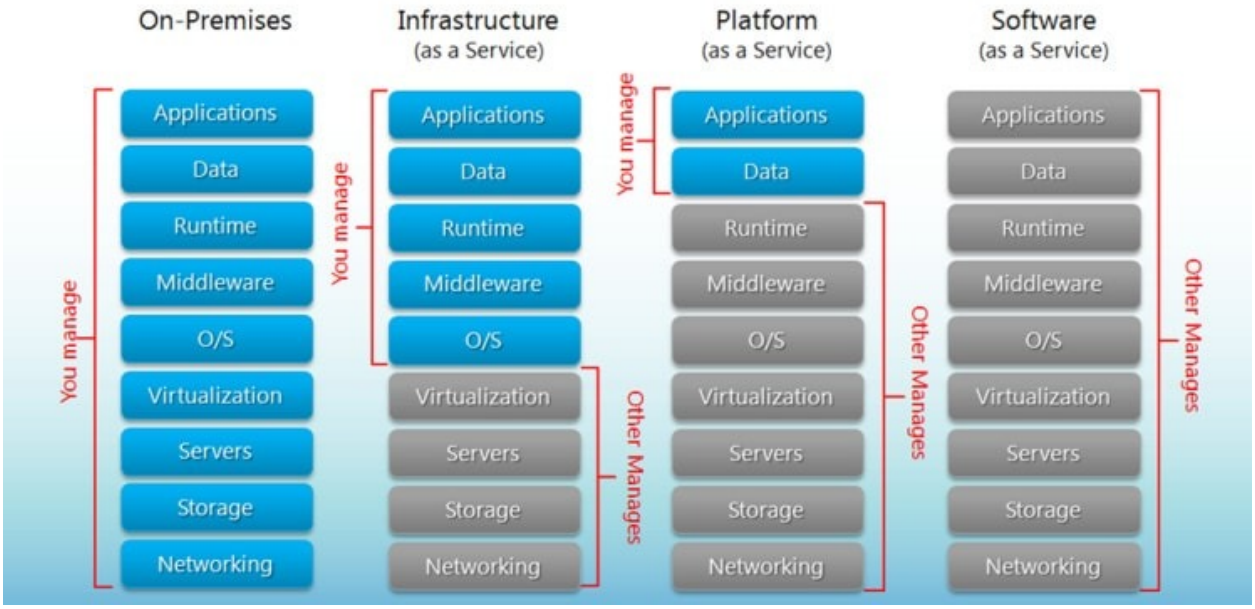

Figure 2: IaaS vs PaaS vs SaaS - image source https://www.bigcommerce.com/blog/saas-vs-paas-vs-iaas/

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 on clouds

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)

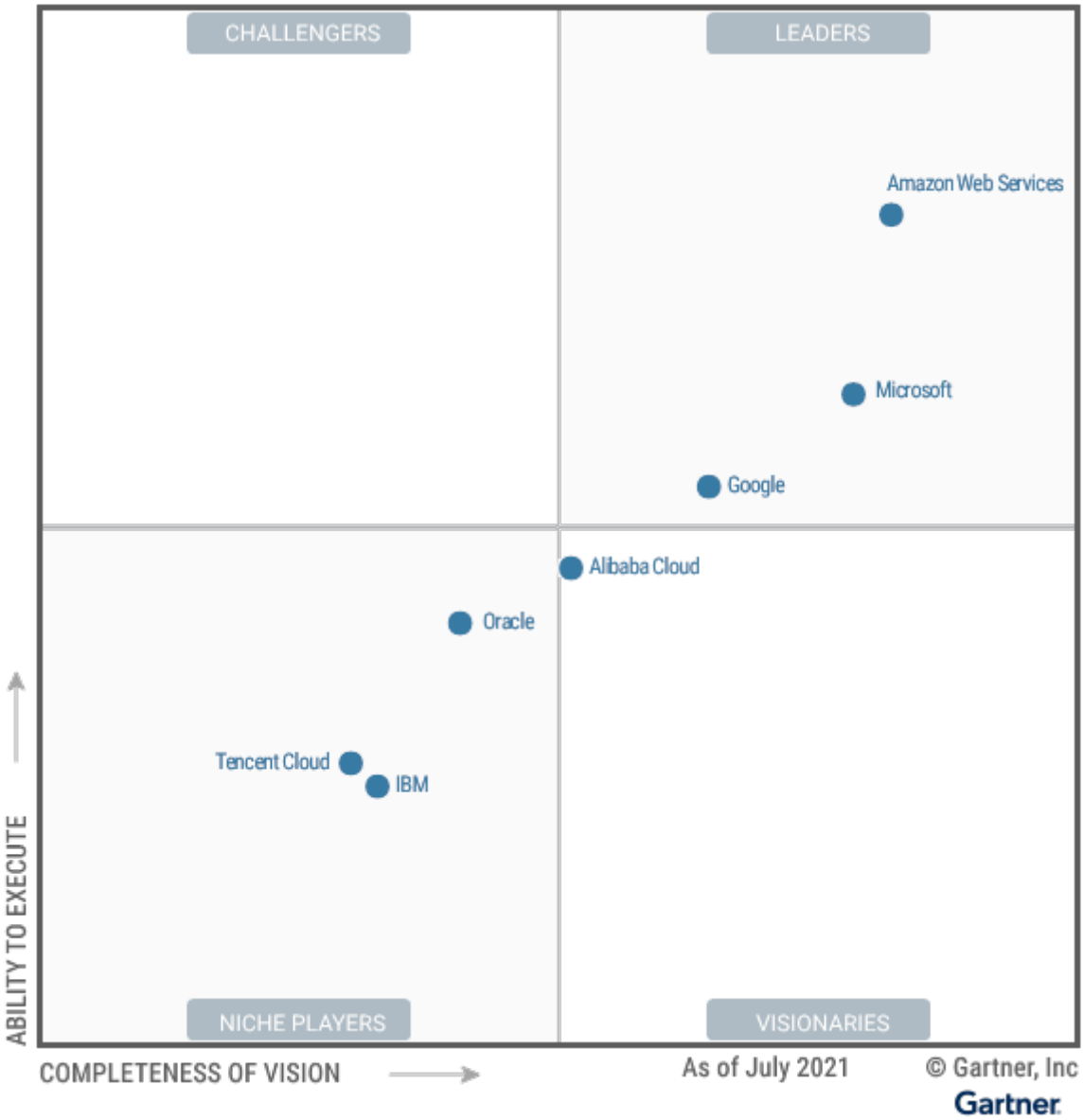

Figure 3: Magic Quadrant for Cloud Infrastructure and Platform Services

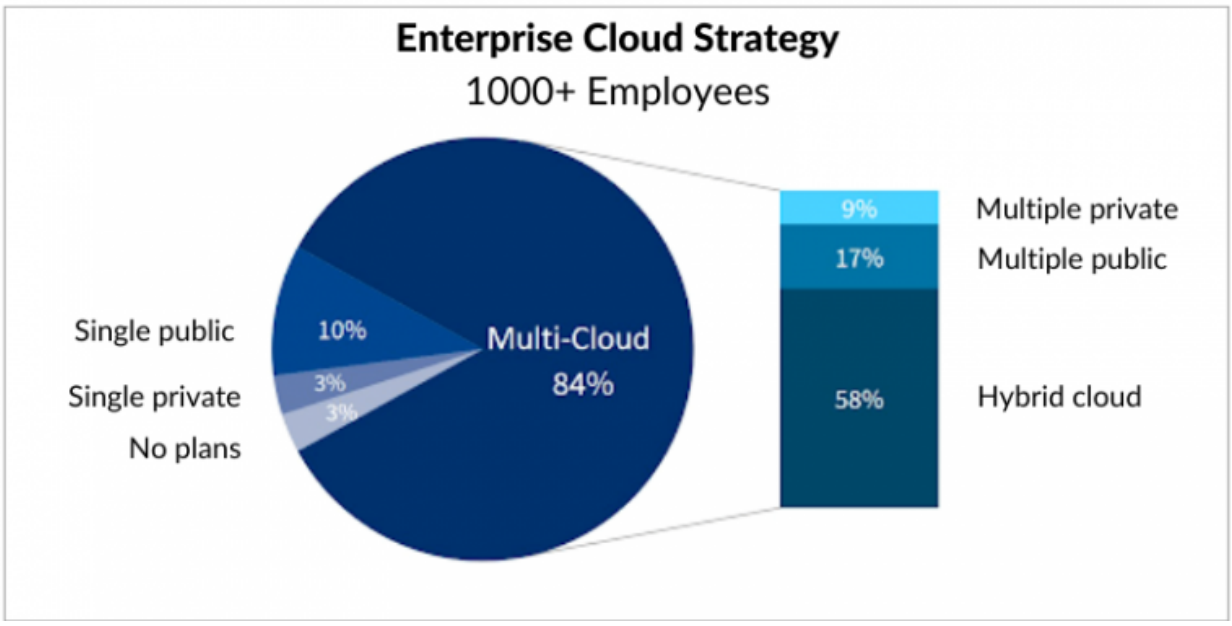

Figure 4: Cloud Strategy - image source Flexera

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 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 to cloud 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

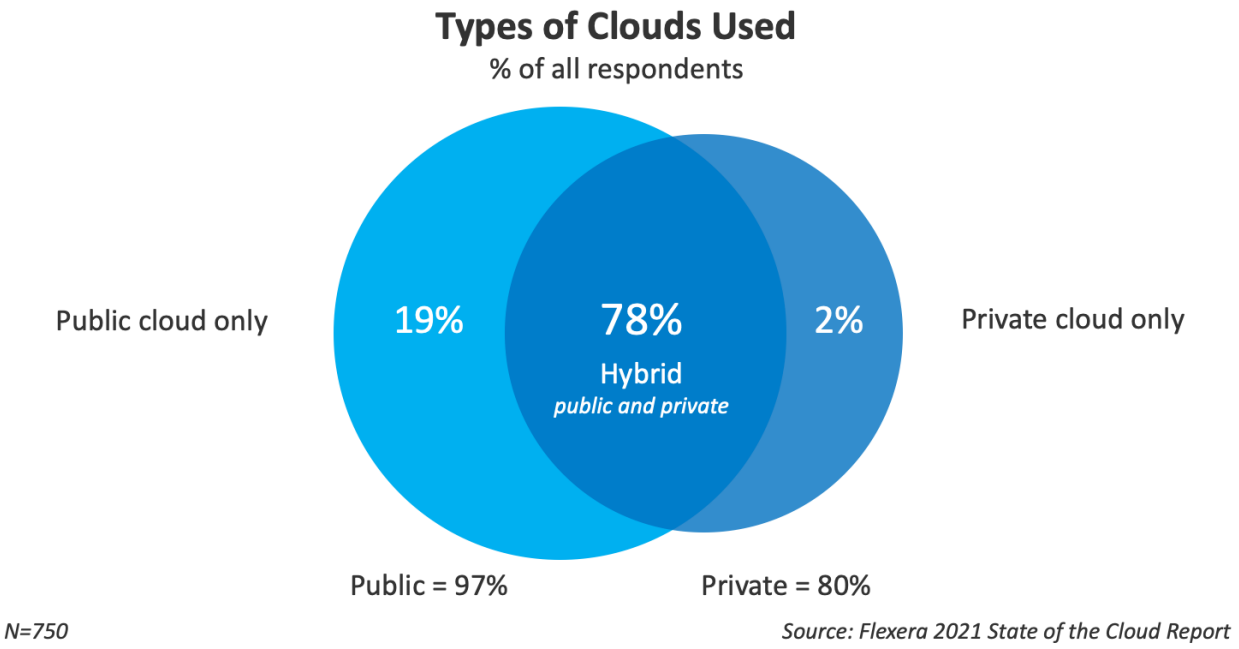

Figure 5: Cloud Strategy - image source Flexera

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 - 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.

1.4 Use Case (R Case)

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

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)

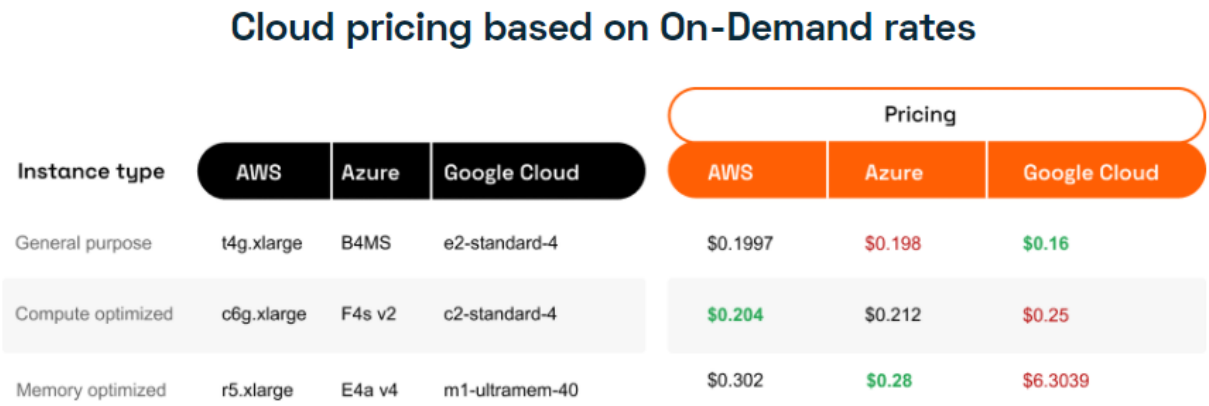

Figure 7: Pricing comparison - image source cast.ai

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. 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
```

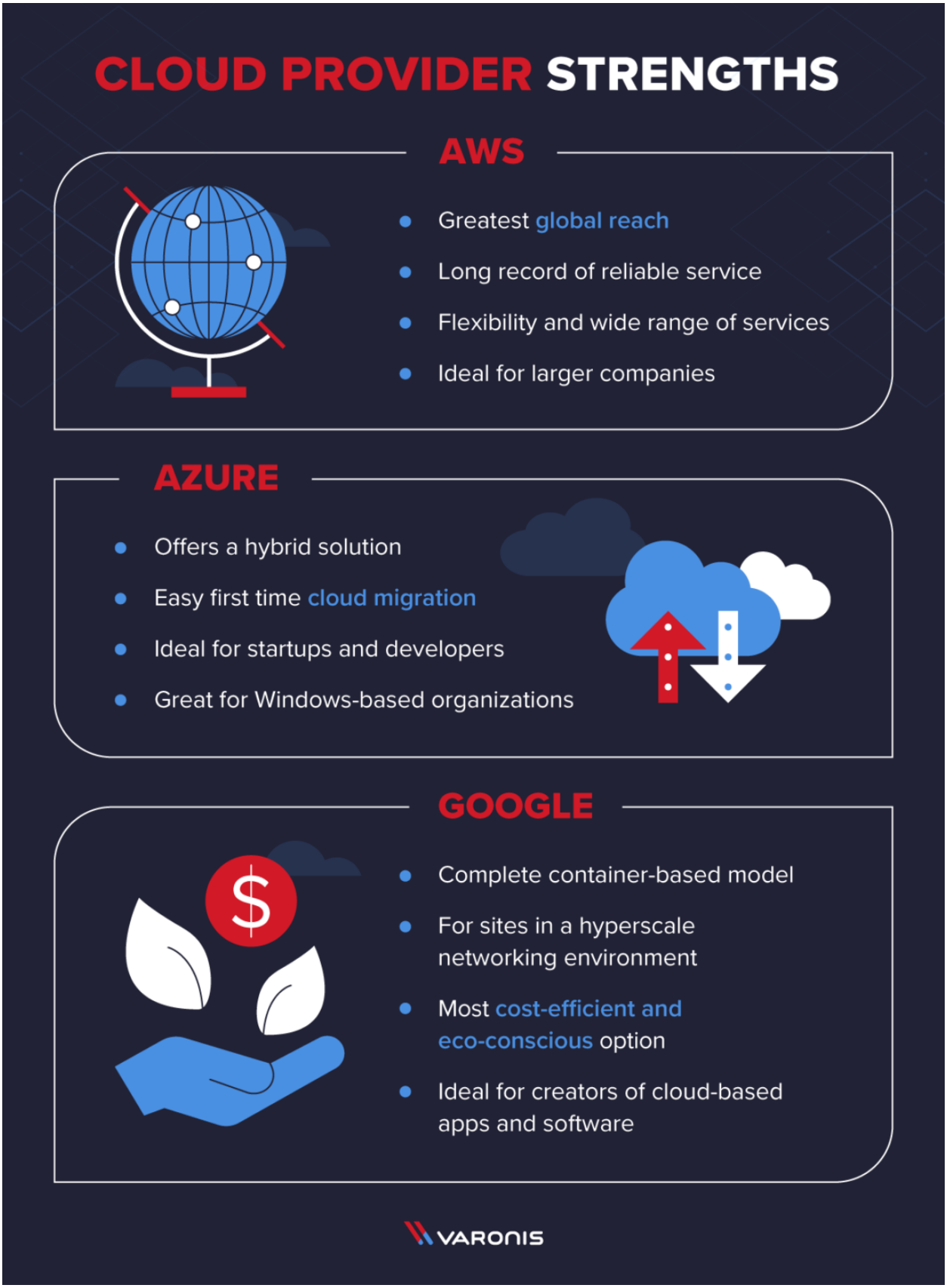

Figure 8: Conclusion - image source www.varonis.com

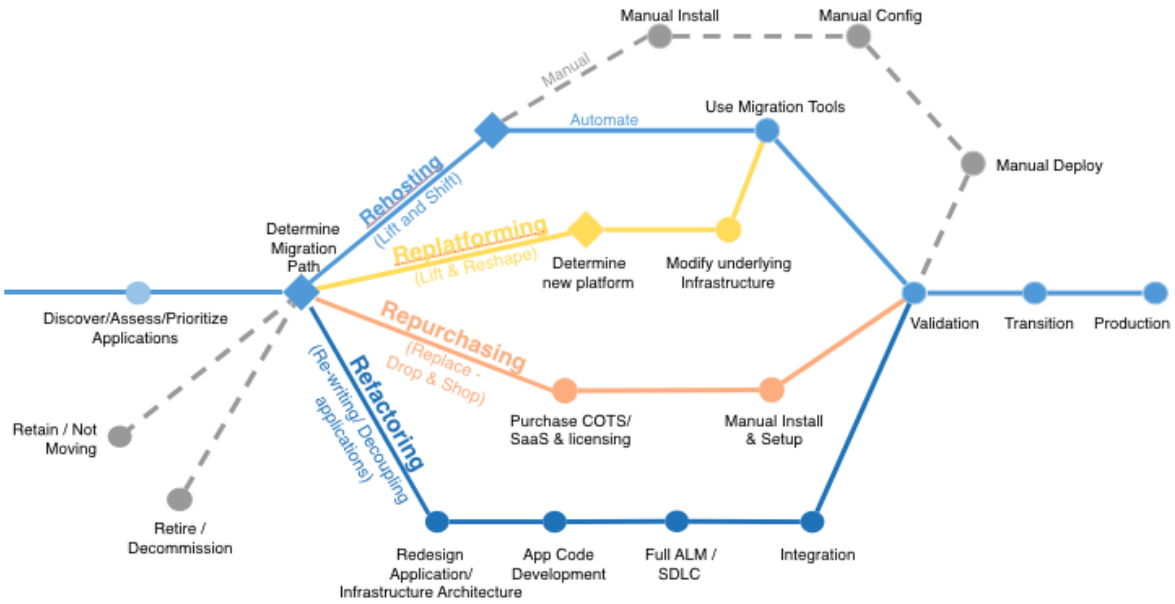

Figure 9: Cloud migration strategy - image source aws.amazon.com

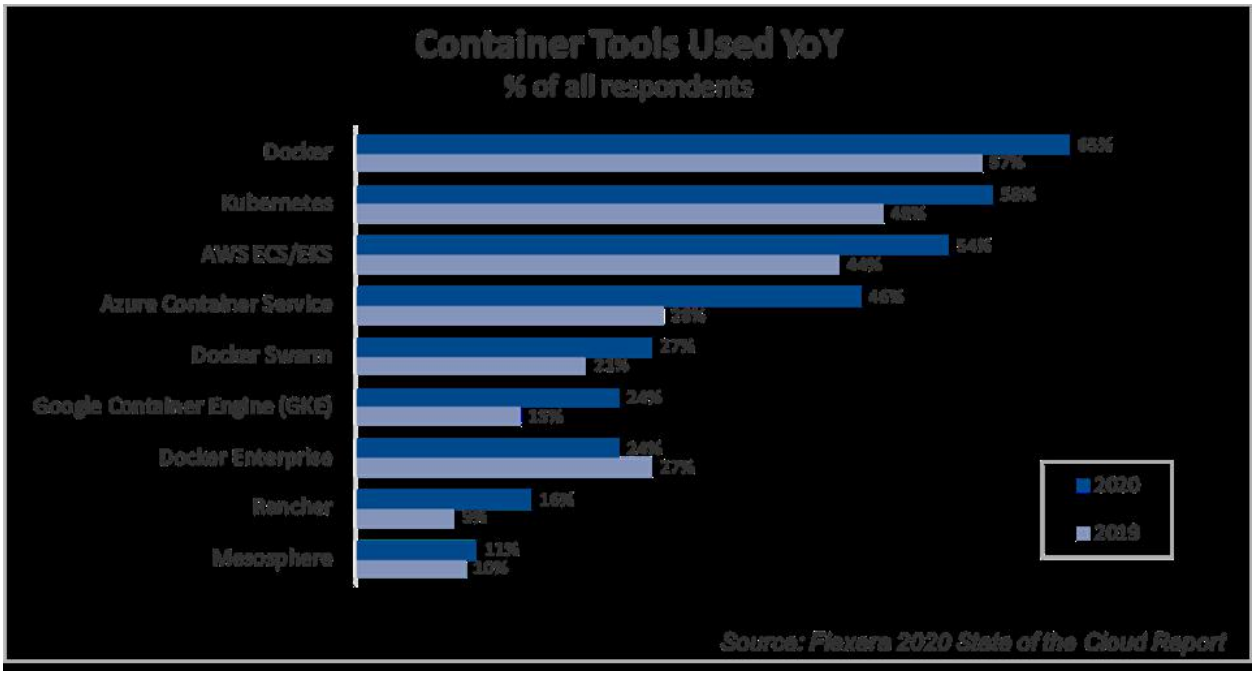

Figure 10: Cloud Strategy - image source Flexera

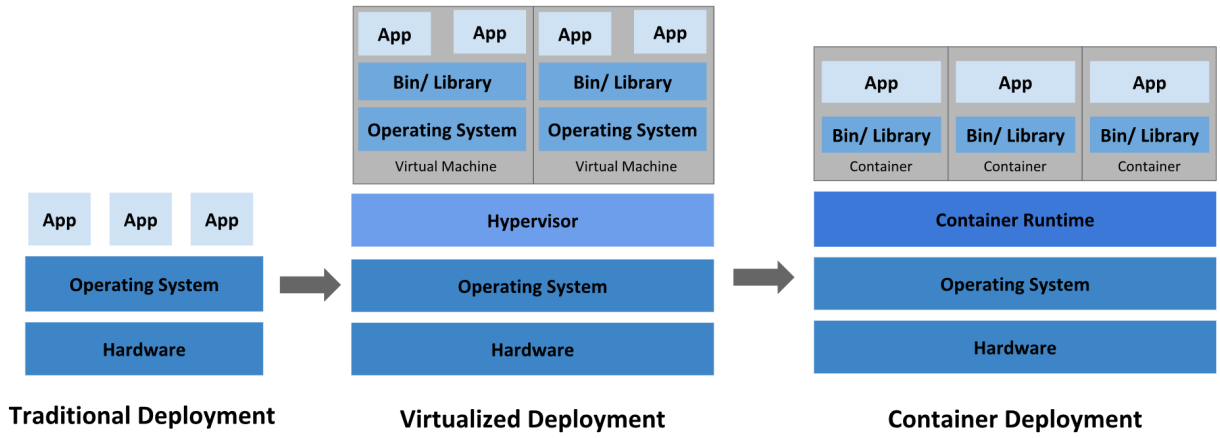

Figure 11: Container evolution - kubernetes.io

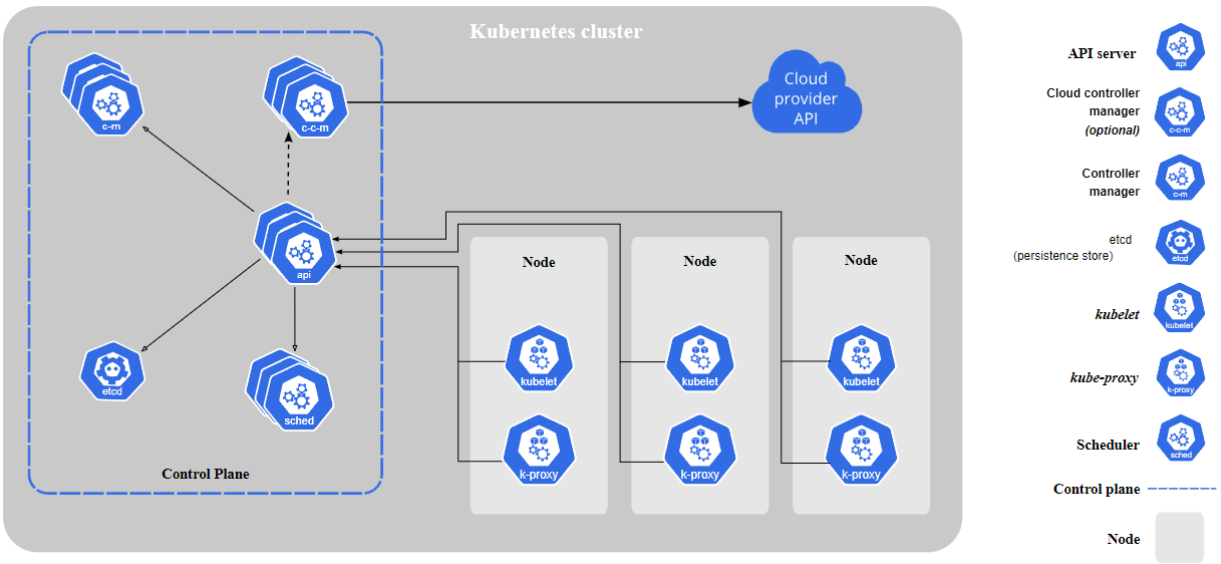

Figure 12: The components of a Kubernetes cluster - kubernetes.io

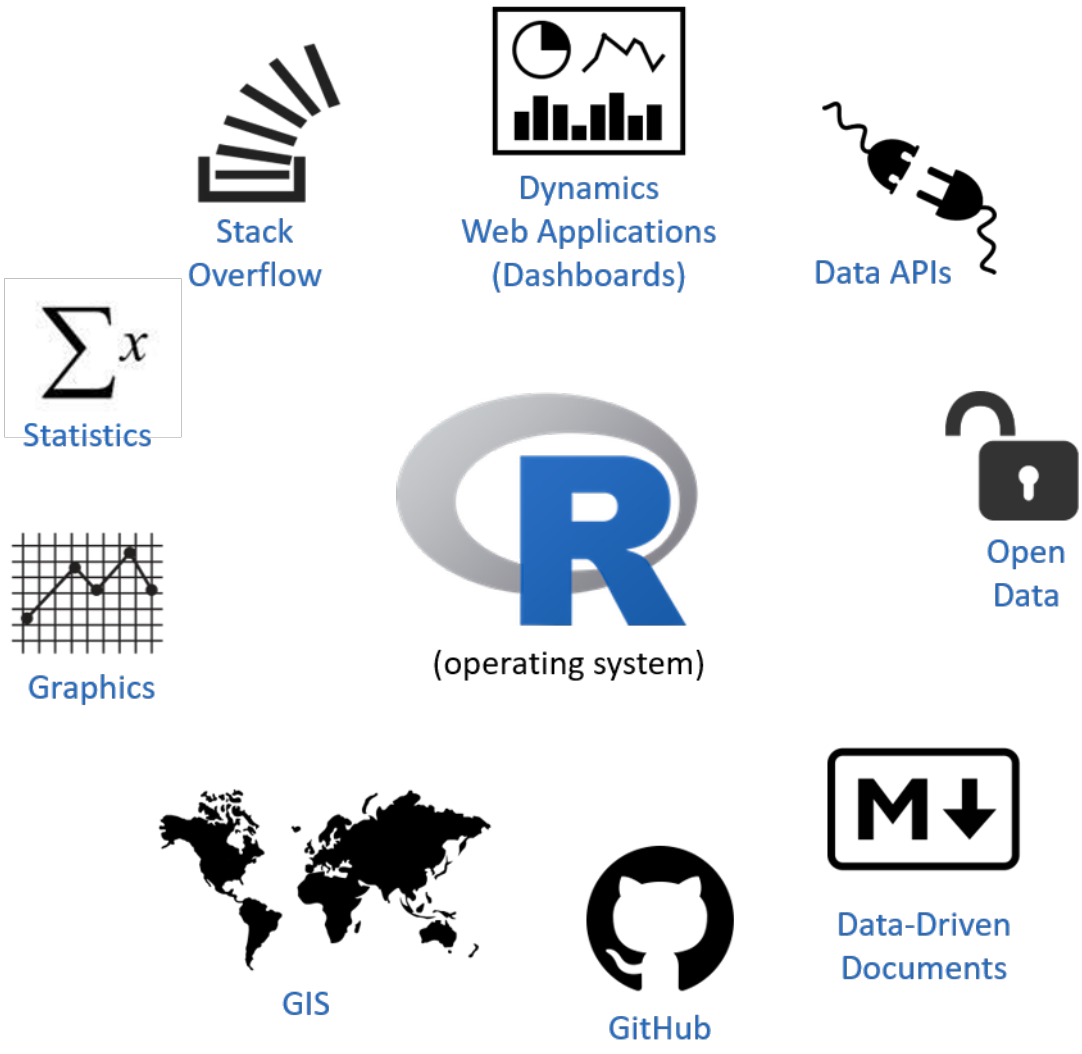

Figure 13: R applications - image source ds4ps.org

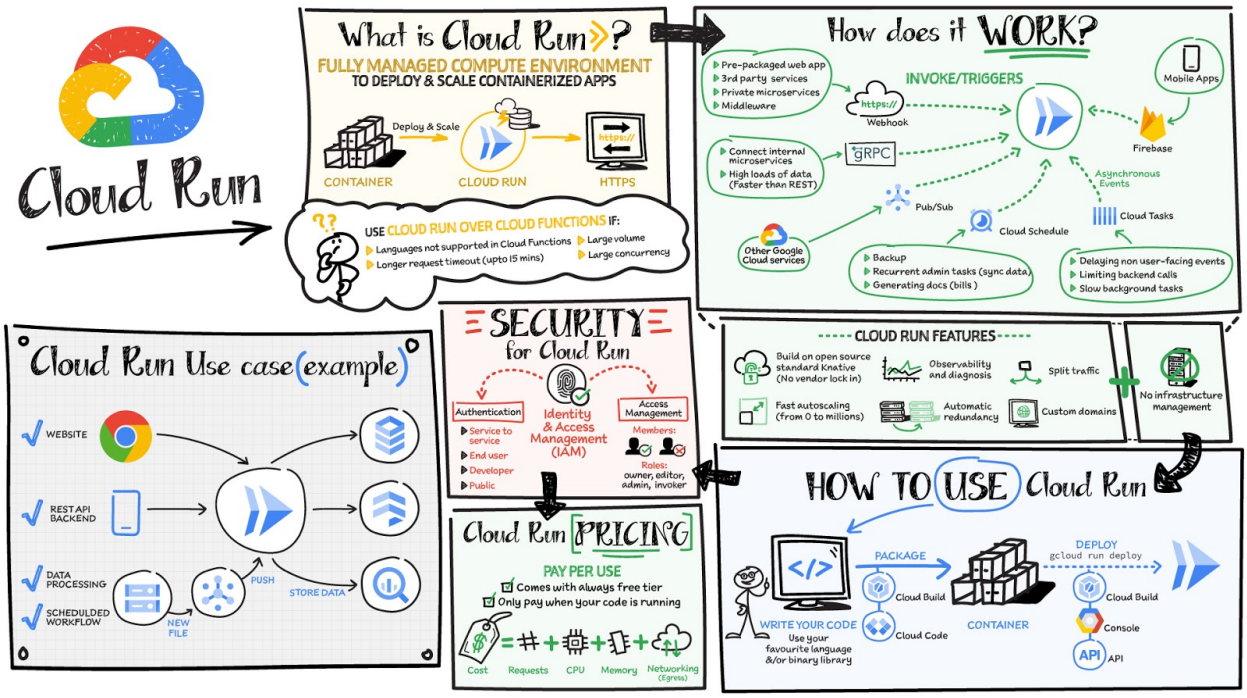

Figure 14: Cloud Run - cloud.google.com

```
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"]
```

Some of the shiny server nuances require to make sure that the logs directory will be created with relevant permissions. Therefore we are going to execute a simple bash script during the container building process.

```
#!/bin/sh
mkdir -p /var/log/shiny-server
chown shiny.shiny /var/log/shiny-server
exec shiny-server >> /var/log/shiny-server.log 2>&1
```

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

It is easy to monitor and track deployments in Google Cloud Console interface. Moreover, deployments can be fully automated via terraform and additional monitoring measures could be made in order to ensure correct application behavior.

Application is already deployed and now can be reached via this URL https://shinydashboard-k6i6icomba-lz.a.run.app

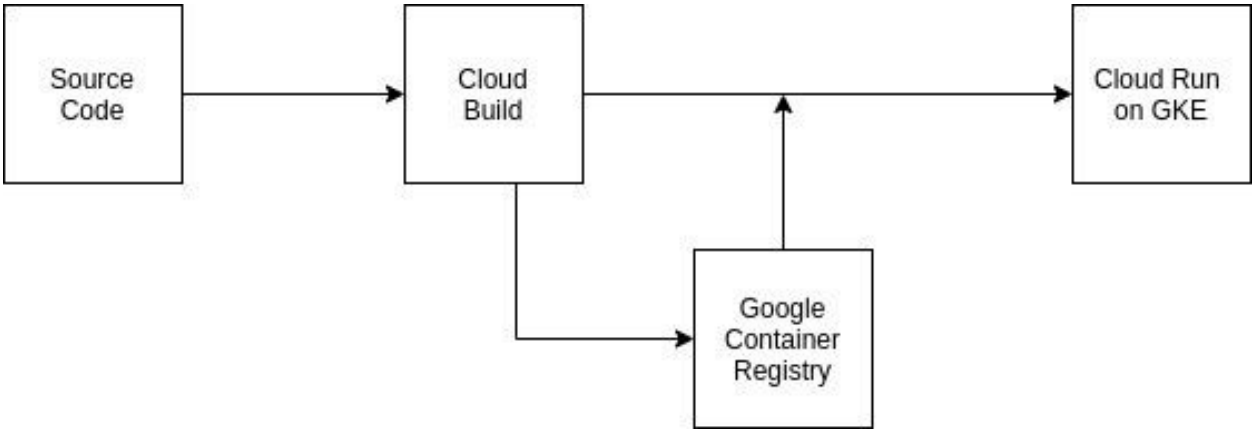

Figure 15: Cloud Run Process - image source https://medium.com/google-cloud

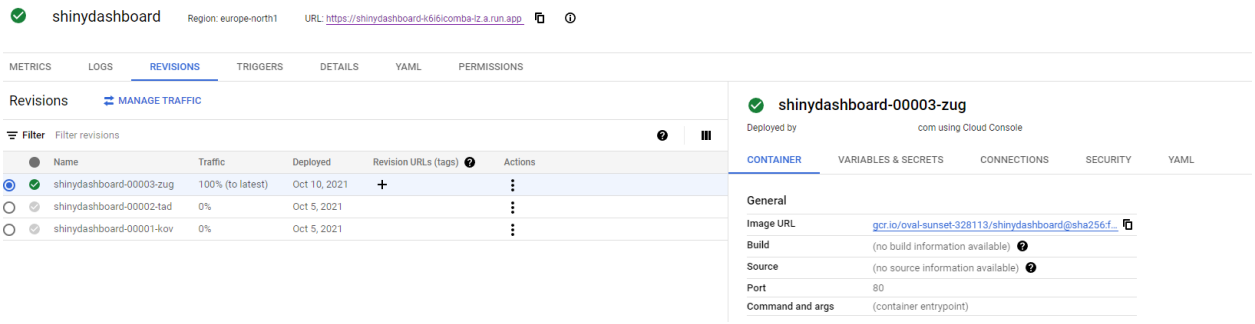

Figure 16: Cloud Run Deployments

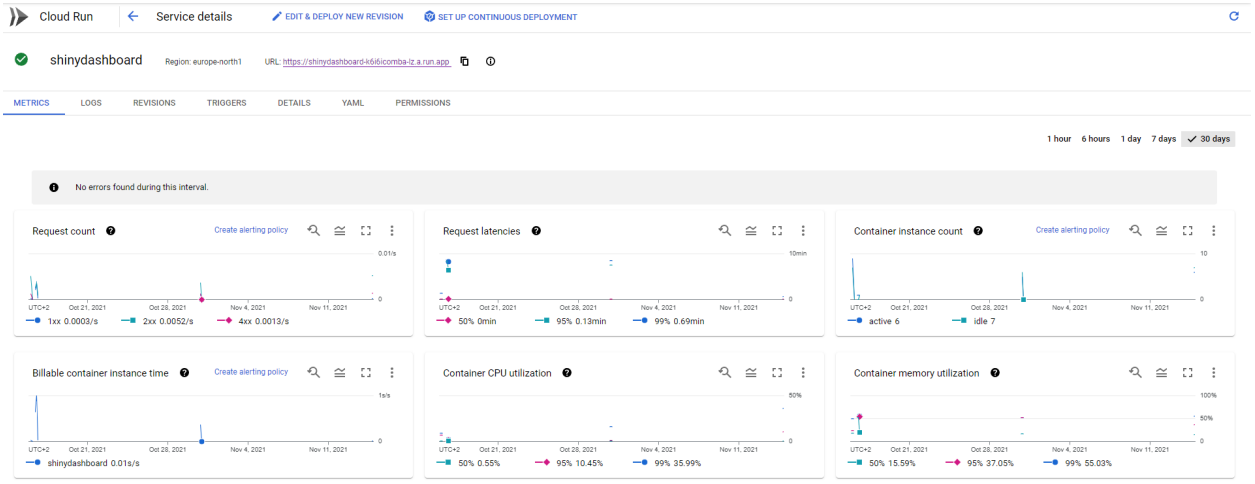

Figure 17: Cloud Run Metrics

2 Public Cloud Setup

2.1 Security

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.

Despite bringing many benefits, the cloud computing paradigm imposes serious concerns in terms of security and privacy, which are considered hurdles in the adoption of the cloud at a very large scale. Customers and organizations in the cloud should be aware of threats, attacks and vulnerabilities, as security awareness is considered the first step to ease the adoption of the cloud

https://www.mdpi.com/2076-3417/11/19/9005/pdf

Various researches have been done on privacy and security in CC. Arjun et al in paper [3] stated that security issues are based on the cloud provider, service user, and instance. Another researcher in paper [4] argued that security issues are based on the delivery model, PaaS, IaaS, and SaaS.

https://iopscience.iop.org/article/10.1088/1742-6596/1979/1/012038/pdf

As in cloud computing, oceans of data will be stored. Data stored in public cloud would face both outside attacks and inside attacks since public cloud provider themselves are untrusted.

https://ieeexplore.ieee.org/document/8622531

A. Validation of OTP B. Integrity Checking C. Access Control D. Secure Deletion E. Encryption F. Data Masking G. Intrusion Detection System

https://www.ijrte.org/wp-content/uploads/papers/v8i1s4/A10030681S419.pdf

Almost three-quarters of organizations hosting data or workloads in the public cloud experienced a security incident in the last year. Seventy percent of organizations reported they were hit by malware, ransomware, data theft, account compromise attempts, or cryptojacking in the last year. Ì Data loss/leakage is the number one concern for organizations. Data loss and leakage topped our list as the biggest security concern, with 44% of organizations seeing data loss as one of their top three focus areas. Ì Ninety-six percent of organizations are concerned about their current level of cloud security. Data loss, detection and response, and multi-cloud management top the list of the biggest concerns among organizations. Ì Multi-cloud organizations reported more security incidents in the last 12 months. Seventy-three percent of the organizations surveyed were using two or more public cloud providers and reported more security incidents as those using a single platform. Ì European organizations may have the General Data Protection Regulation (GDPR) to thank for the lowest attack rates of all regions. The GDPR guidelines' focus on data protection, and well-publicized ransomware attacks have likely led to these lucrative targets becoming harder for cybercriminals to compromise in Europe. Ì Only one in four organizations see lack of staff expertise as a top concern despite the number of cyberattacks reported in the survey. When it comes to hardening security postures in the cloud, the skills needed to create good designs, develop clear use cases, and leverage third-party services for platform tools are crucial but underappreciated. Ì Two-thirds of organizations leave back doors open to attackers. Accidental exposure through misconfigurations continues to plague organizations. Security gaps in misconfigurations were exploited in 66% of attacks (either through attackers exploiting a flaw in the web application firewall to access account credentials or attackers taking advantage of a misconfigured resource), while 33% of attacks used stolen credentials to get into cloud provider accounts

https://secure2.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/sophos-the-state-of-cloud-security-2020-wp.pdf

Zero trust https://www.researchgate.net/publication/341806714_Establishing_a_Zero_Trust_Strategy_in_Cloud_Computing_Environment

https://www.juniper.net/content/dam/www/assets/white-papers/us/en/security/the-rise-of-zero-trust.pdf

https://media.defense.gov/2021/Feb/25/2002588479/-1/-1/0/CSI_EMBRACING_ZT_SECURITY_MODEL_UOO115131-21.PDF

2.2 Networking

https://secure2.sophos.com/en-us/medialibrary/Gated-Assets/white-papers/sophos-the-state-of-cloud-security-2020-wp.pdf

For example, a misconfigured route table on an organization's firewall leaves the window open. Virtual machines running private server workloads or hosting sensitive data suddenly become accessible from the internet.

The impact of configurations on data security Organizations had databases open to the Interne

Some security recommendations for network security can be summarized as follows: • The internal communication of the cloud must adopt secure communication techniques such as HTTPS, and also the transmission channel must be encrypted by TLS. • Using anomaly detection solutions for HTTP requests that can effectively prevent any malicious network intrusion behaviors. • The cloud can use public security services such as web application firewalls (WAF), virtual firewalls, virtual bastion machines, virtual host protection and virtual database audit systems

https://www.researchgate.net/publication/341806714_Establishing_a_Zero_Trust_Strategy_in_Cloud_Computing_Environment

https://cloud.google.com/kubernetes-engine/docs/concepts/network-overview

2.3 Infrastructure as Code

Terraform 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. Terraform's infrastructure-as-code (IaC) approach supports DevOps best practices for change management, letting you manage Terraform configuration files in source control to maintain an ideal provisioning state for testing and production environments.

https://cloud.google.com/architecture/managing-infrastructure-as-code

2.4 Use Case

2.4.1 Shiny Server on Compute Engine

main.tf:

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

// 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"]
 }
}
```

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

```
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.
```