Tuomas Autio

Securing a Kubernetes Cluster on Google Cloud Platform

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

19 April 2021

Author Title Number of Pages Date	Tuomas Autio Securing a Kubernetes Cluster on Google Cloud Platform 32 pages 19 April 2021
Degree	Bachelor of Engineering
Degree Programme	Information and Communication Technology
Professional Major	Smart Systems
Instructors	Aarne Klemetti, Researching Lecturer Kalle Sirkesalo, Director of Platform

The goal of this thesis was to investigate possible security problems when creating cloud infrastructure with a small budget that was set. Optimizing the cost of creating the cloud environment was also looked into. Infrastructures made in the cloud environment can be vulnerable if not made properly.

The thesis was commissioned by a DevOps-oriented software company called Eficode Oy. Eficode is currently active in six countries and employs over 300 workers. The work was done as a consulting job for a customer who will be referred to as Customer x in this thesis.

The thesis investigates the relationship between the budget and security in a cloud environment and whether one has a significant effect on the other. The study is based on a literature review and other academic papers. The opinions presented in different research papers varied between the two options. This study and some papers however argue that security does not need to be expensive. Most of the cloud hosting platforms provide a best practices guide on how to secure and cost optimize the platform, which had a high impact on the outcome of this study.

The goal that was set was reached and the defined budget was satisfied.

Keywords	Kubernetes, Security, Docker, GCP

Tekijä Otsikko Sivumäärä Aika	Tuomas Autio Securing a Kubernetes Cluster on Google Cloud Platform 32 sivua 19.4.2021
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja Viestintätekniikka
Ammatillinen pääaine	Smart Systems
Ohjaajat	Tutkijaopettaja Aarne Klemetti Director of Platform Kalle Sirkesalo

Insinöörityön tavoitteena oli luoda pilviympäristöön infrastruktuuri johon asiakkaan vanha ympäristö tultaisiin siirtämään. Samalla lähdettiin tutkimaan onko budjetilla sekä tietoturvalla yhteyksiä toisiinsa ja kuinka suuri vaikutus niillä on toisiinsa.

Insinöörityö tuotettiin projektina Eficodelle joka DevOpsiin orientoitunut ohjelmisto yritys Suomessa. Eficode työllistää tällä hetkelle yli 300 työntekijää kuudessa maassa.

Käymme myös pintaraapaisulta läpi Kubernetestä, Dockeria sekä Terraformia koska ne toimivat tämän insinöörityön perustana. Insinöörityössä käydään muun muassa läpi Kuberneteksen servicejä, Docker konttien ja normaalin virtuaalikoneen eroja sekä miten infrastruktuuria voidaan säilyttää koodin muodossa.

Työssä käytettiin hyväksi paljon pilvipalvelu tarjoajien kirjoittamia suositeltuja käytäntöjä, joista saa hyvää osviittaa miten hyvät ja turvalliset pilviympäristöt luodaan. Myös kustannusten optimointia katsottiin tarkemmin ja käytettiin hyväksi. Näillä oli suuri vaikutus insinöörityön lopputulokseen.

Avainsanat	Kubernetes, Tietoturva, Docker, GCP

Contents

List of Abbreviations

Docker	Platform to enable shipping and running applications quickly
GCP	Google Cloud Platform
GKE	Google Kubernetes Engine
HCL	Hashicorp Configuration Language
HTTPS	Hypertext Transfer Protocol Secure
IaC	Infrastructure as Code
IaaS	Infrastructure as a Service
ICMP	Internet Control Message Protocol
NAT	Network Address Translation
OS	Operating System
SSH	Secure Shell Protocol
TLS	Transport Layer Security
VM	Virtual Machine
VPC	Virtual Private Cloud
VPN	Virtual Private Network
YAML	Yet Another Markup Language

1 Introduction

Even though many services are moving towards the cloud nowadays, it is still a bit bitter to many. The enormous bills of cloud services, when done wrong, scare off users. This thesis will explore the many ways of creating a working and secure Kubernetes cluster in the cloud with a budget in mind. In this project, investigating was done to find answers to the following questions: How are costs cut in cloud services? Does cutting costs affect the security of the service?

The project was commissioned by a DevOps-oriented software company called Eficode Oy. Eficode's customer who will be referred to as Customer X used to run their production environment in a cloud-hosted Rancher instance. The purpose of this project was to migrate the production environment to the Google Cloud Platform. The biggest reason for this change was that the version of Rancher the customer had in place was no longer getting security updates. Because Customer X is working in the medical field, it was essential to ensure all the necessary security-related actions were taken.

1.1 Technical objectives

The main objective was to create a working and secure version of the Rancher environment for the Google Cloud Platform. The traffic between the web application and the SQL database needed to be secured and the database could only be accessed by authorized users and applications.

1.2 Financial objectives

Costs also needed to be considered, which might impact the task of securing the environment because many security features are chargeable. Most of the current nuisance related to cloud services comes from the "high" price of the bill. The possible solutions for these problems will discussed in this thesis.

2 Planning

2.1 Infrastructure

The planning phase of the infrastructure was simple because an existing plarform had been already made that just needed to be transferred to Google Cloud Platform (GCP). A container management system was needed to run the dockerized microservices and a database to hold the data. In the following chapter, a closer look will be taken at GCP's methods and services.

2.2 CloudSQL

CloudSQL is a database service offered and hosted in GCP. Google's CloudSQL offers fully managed MySQL, PostgreSQL, and SQL Server databases. CloudSQL also offers built-in functionality that ensures high availability and reliability. Quickly configurable replication and backups ensure safety in unpleasant scenarios [1.]

The downsides of using a Google-maintained database are trust and confidentiality. How can the users be sure that data is not leaked or modified by the maintainer? The database might return false data because of a defect in the system, or the data integrity might suffer[2.]

2.3 Kubernetes

Kubernetes is an open-sourced container management platform initially developed by Google and later continued by Cloud Native Computing Foundation [3.] Kubernetes is filled with built-in features such as secret management, service discovery, load balancing, self-healing, and automated rollbacks. Kubernetes is highly customizable so that the features can be disabled upon demand.

Kubernetes is highly used in the industry because of its many benefits. It enables faster scaling and updating of applications in a DevOps mindset manor. Portability, easy deployment, near-zero downtime, flexibility, secret management, among other features, bring a set of tools to enable the user to do whatever they imagine.

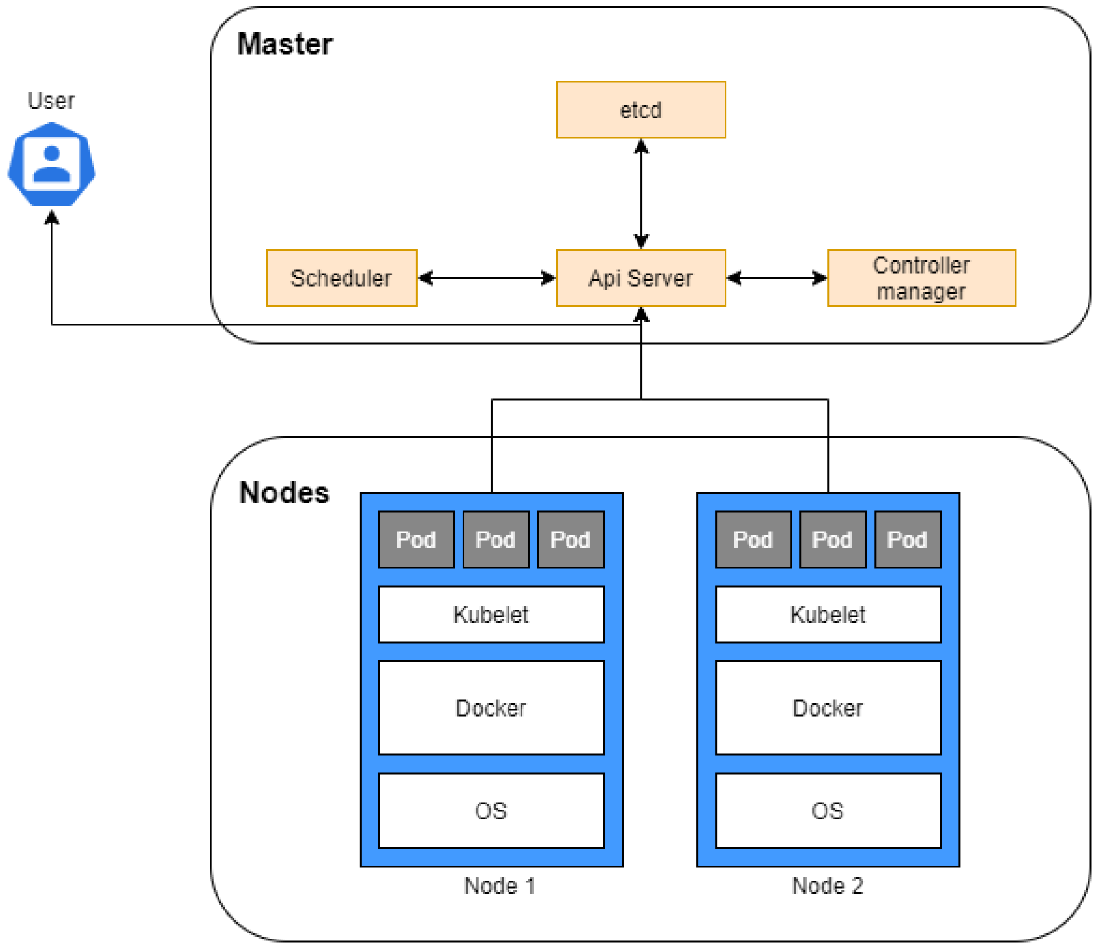


Figure 1. Kubernetes architecture. based on data from Kubernetes website [3].

Docker is not the only container runtime used in nodes. Some might prefer containerd or rocket. Different parts of Kubernetes are controlled by the control plane, the core of which is the API server. The API server can be controlled through command-line interfaces such as *kubectl* or *kubeadm*. The API lets the user query and control, for example, the Pods and Namespaces in Kubernetes. The architecture is illustrated in figure 1.

The Control plane also has a scheduler, etcd, and controller manager. The scheduler watches pods and nodes and makes scheduling decisions based on different metrics that include constraints and deadlines. Etcd is the server that contains all the cluster data. The controller manager contains all the controllers for nodes, endpoints, replications, tokens, and service accounts. Controllers watch the state of the cluster and attempt to make changes to the cluster to reach the desired state.

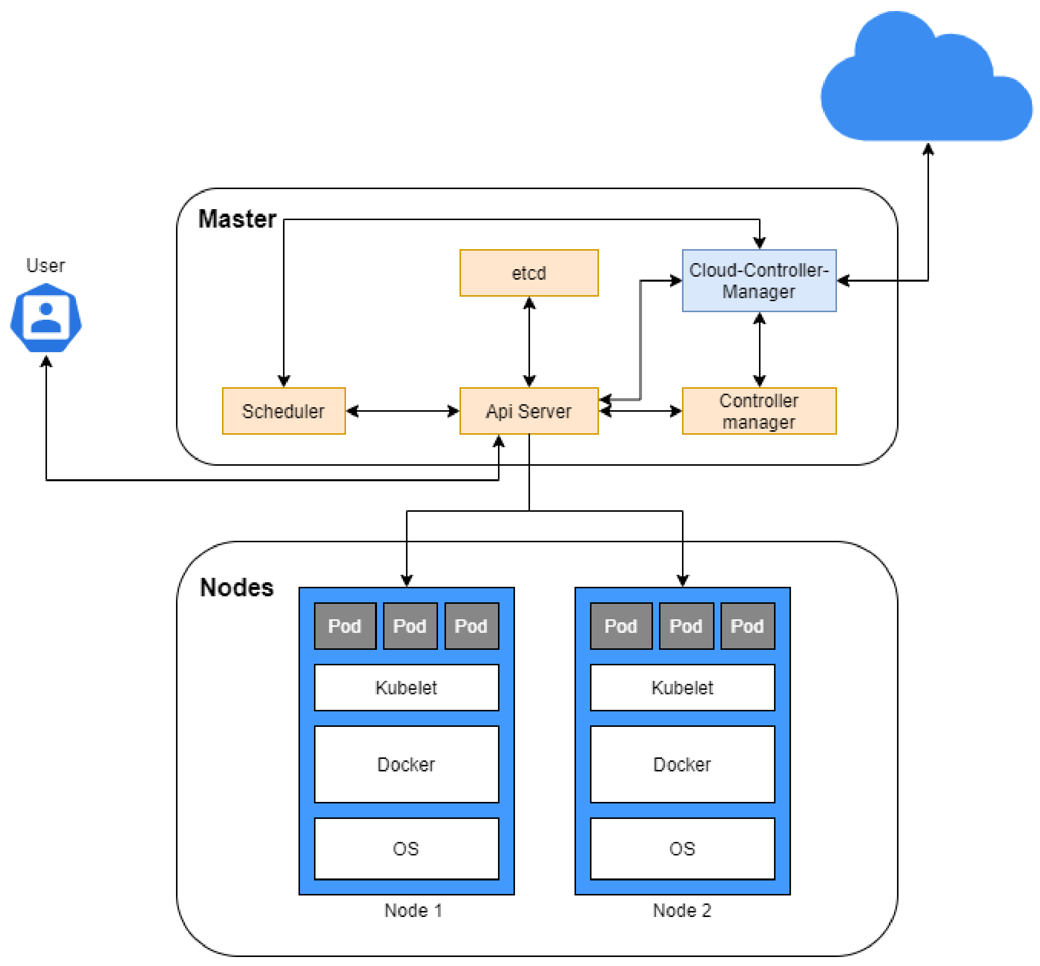


Figure 2. Kubernetes architecture with CCM based on images from Kubernetes website [3].

Cloud-hosted Kubernetes platform usually contains one extra component, which is called Cloud-Controller-Manager (CCM) (illustrated in figure 2). The purpose of the CCM is to link the cloud providers' API with the cluster. That way, the user can interact with the cluster from the cloud provider's dashboard.

Users can run applications inside Kubernetes by defining workloads. Kubernetes comes with built-in workloads like Deployment, Job, or Cronjob. Workloads are usually defined in YAML, which is a markup language. Custom workloads can also be defined but that will not be focused in this thesis.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

Figure 3. Example YAML of a deployment workload

The example workload in figure 3 deploys the Nginx web server with three replicas. Once the workload has been passed to the API server, the controller manager tries to fulfill the desired states defined in the workload. The user is not restricted to using built-in workloads and can define custom resources to fit the desired purposes. Networking in Kubernetes is also handled with similar YAML manifests.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

Figure 4. Example service manifest

In figure 4, a service is defined to expose an application running on port 9376. ClusterIP creates a service for the cluster that can be internally accessed. External access can be allowed through a Kubernetes proxy.

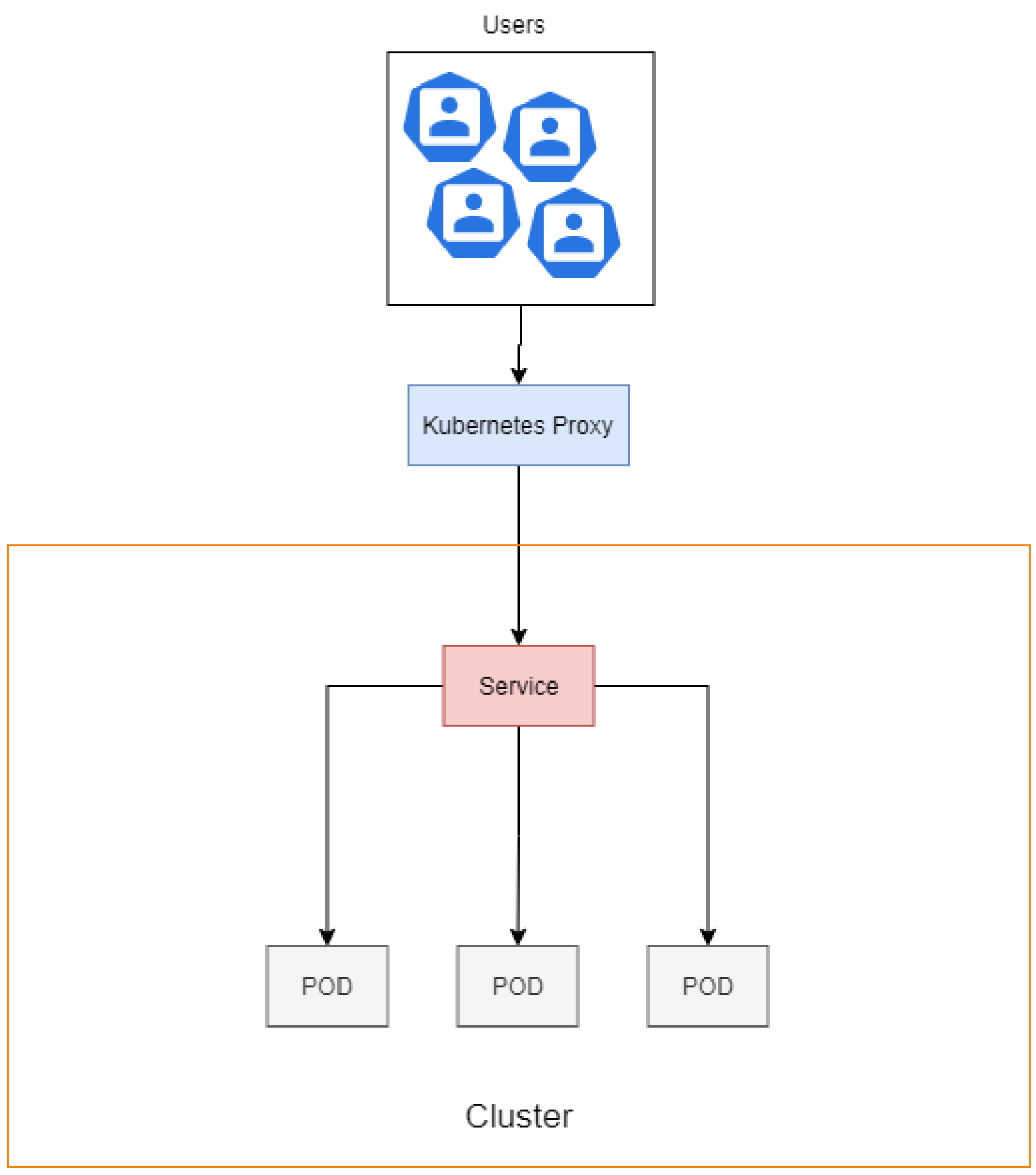


Figure 5. Illustration of ClusterIP solution in a cluster

ClusterIP solution is usually used in a situation when only internal traffic is allowed (see figure 5). It is not the best practice to use this to expose applications to the internet in a production environment.

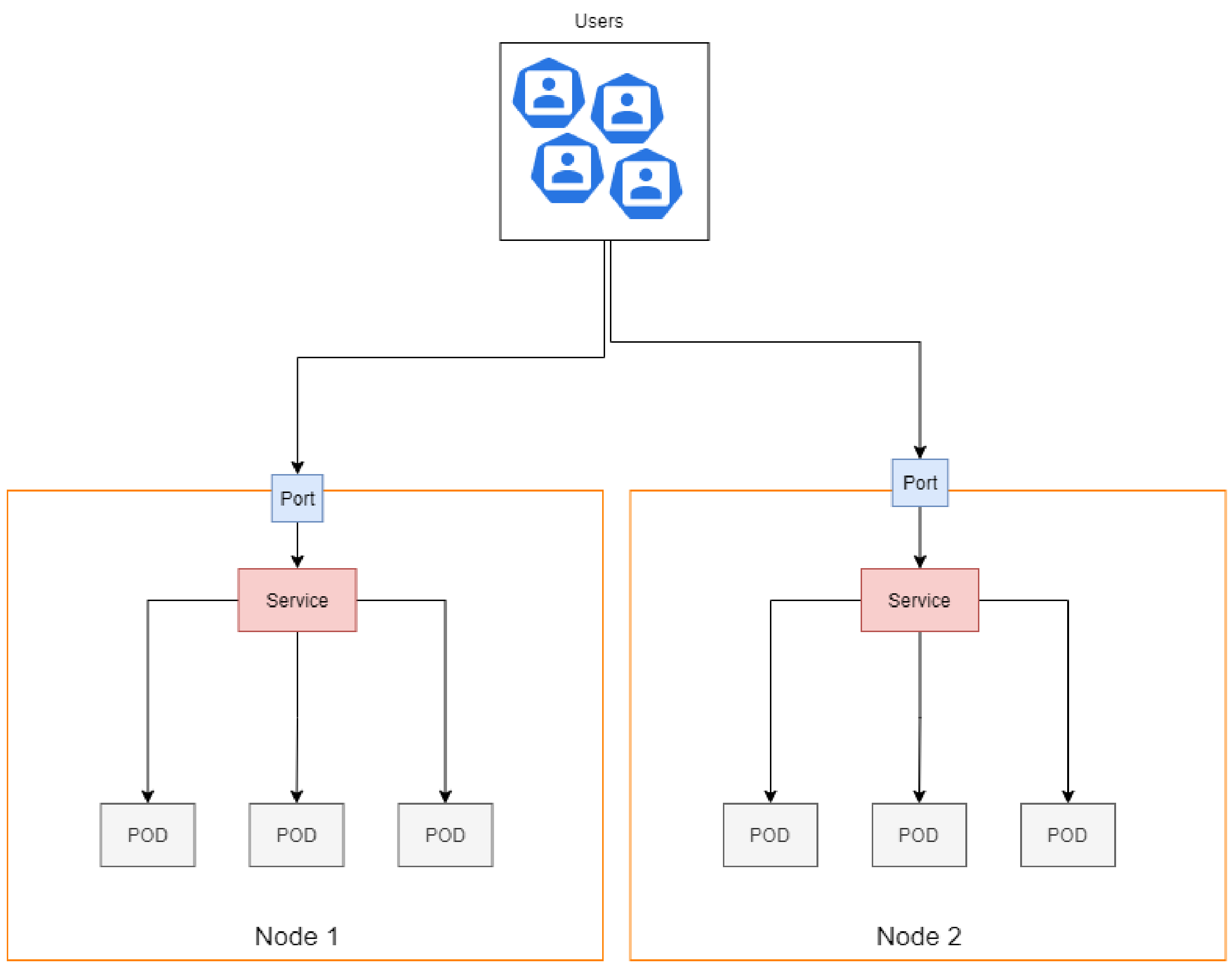


Figure 6. NodePort solution

Other solutions to expose applications are node ports, a load balancer, and an ingress. The node port exposes the service on each node's IP at a static port (as illustrated in figure 6). A ClusterIP service, to which the NodePort service routes, is automatically created [4.]

This method is very cost-efficient but has a few downsides. Only one service per port can lead to many ports being open if many services are needed. From a security perspective, this is considered risky and should always be avoided.

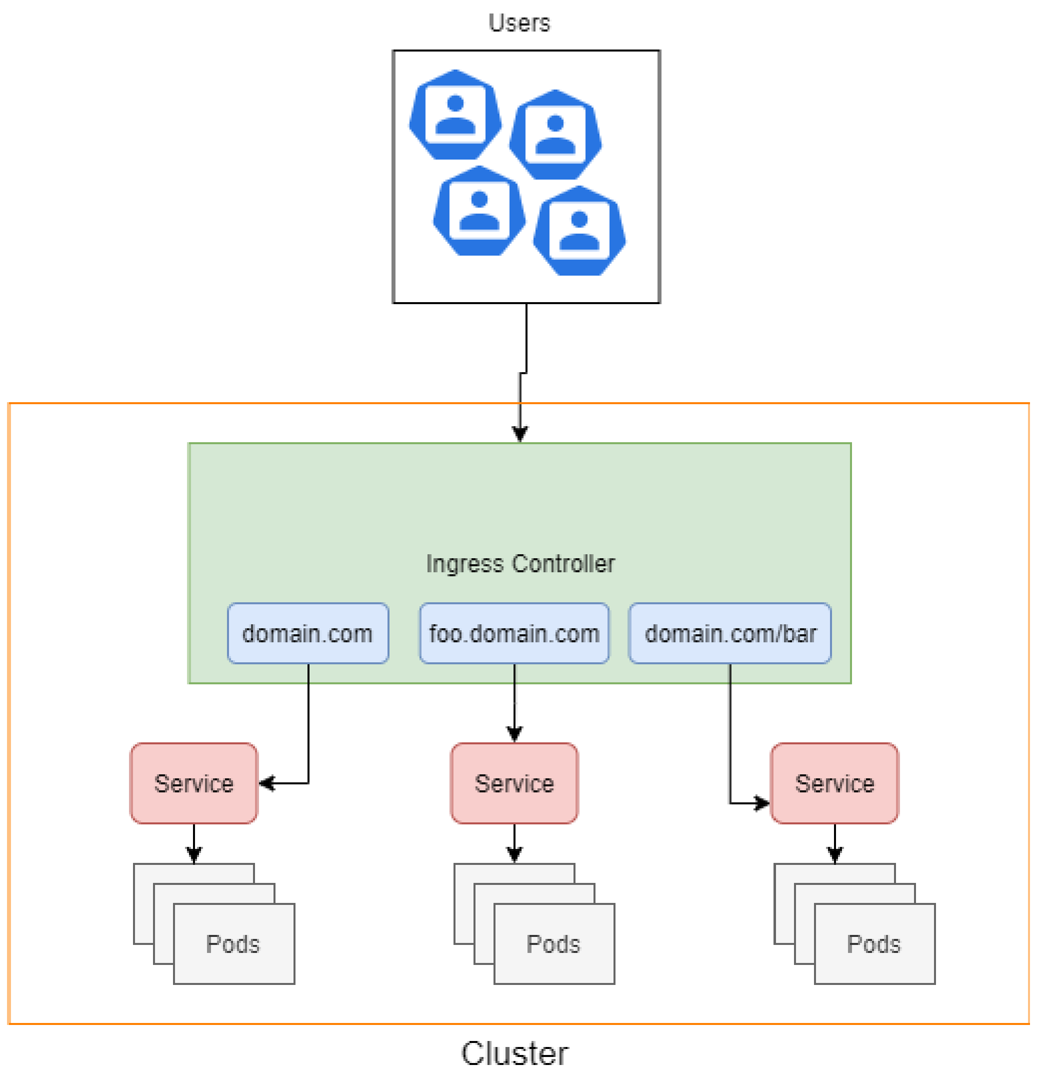


Figure 7. Ingress solution

The ingress is nowadays probably the most used and recommended way to expose large-scale applications in clusters. The ingress is not the most straightforward solution but a versatile one. The ingress itself is not a so-called “service,” but it acts as the load balancer for the services.

The ingress controllers route traffic to the desired services in multiple different ways, path-based and domain-based being the most common (as illustrated in figure 7). There are also different ingress controllers available, like Google Cloud Load Balancer, Nginx, and Istio, to name a few [5.]

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: ingress-example
spec:
  backend:
    serviceName: example-service
    servicePort: 80
  rules:
  - host: domain.com
    http:
      paths:
      - backend:
          serviceName: example-service
          servicePort: 80
  - host: foo.domain.com
    http:
      paths:
      - backend:
          serviceName: foo
          servicePort: 8080
  - host: domain.com
    http:
      paths:
      - path: /bar
        backend:
          serviceName: bar
          servicePort: 8800
```

Figure 8. Example ingress YAML

The traffic can be routed to different services with different ports and define paths in a web server manner (see figure 8). Kubernetes provides many more resources and deployment methods like Helm charts, but those will not be the focus in this thesis.

2.4 Docker

Docker is a virtualization platform that solves the problem “Doesn’t work on my machine.” Instead of running an application in a traditional virtual machine, Docker enables the developer to place the running application in a container that can be deployed and shipped without having problems related to virtual machines. Docker’s architecture is simpler compared to a standard virtual machine [6.]

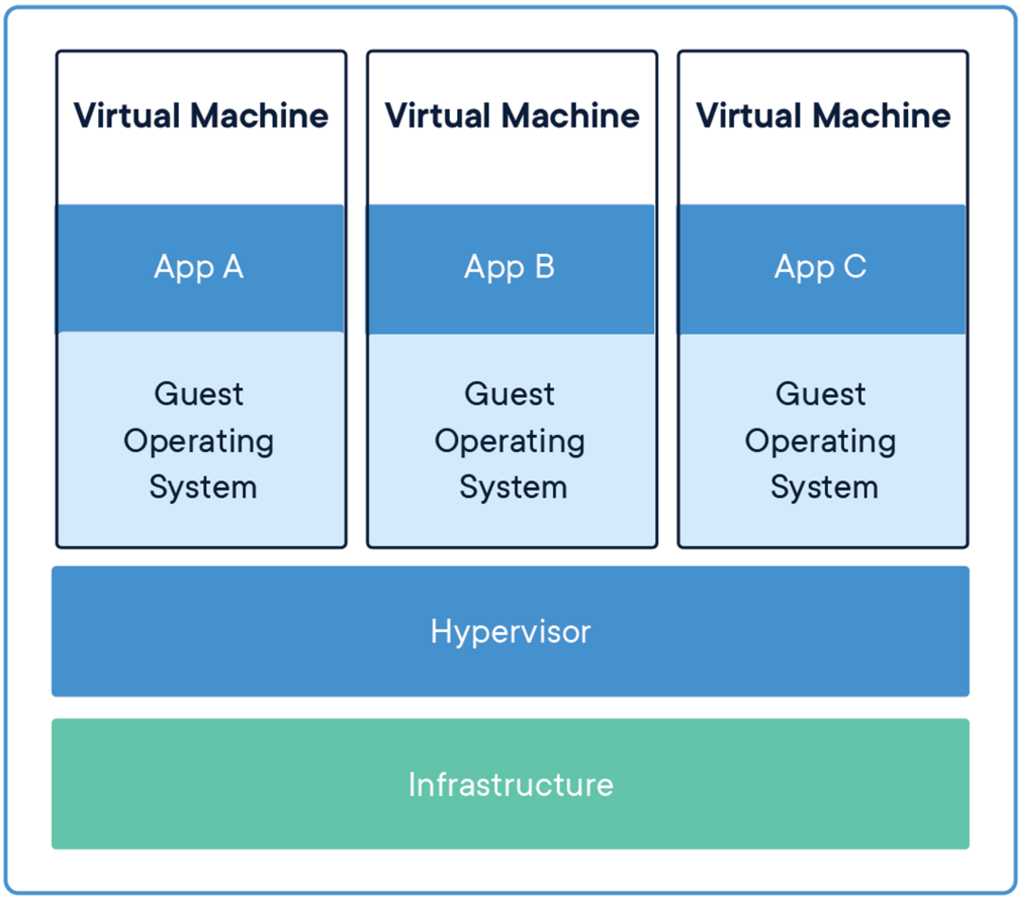


Figure 9. Virtual Machine architecture. image copied from docker website [6].

Applications running on a virtual machine only share the infrastructure and the hypervisor (see figure 9). A hypervisor allows virtual machines to share the resources of the host machine virtually. Each VM has its operating system, which makes them heavier to run compared to containers. In modern-day microservices, having heavy virtual machines run lightweight applications does not meet with best practices.

Running applications in virtual machines might also lead to conflicts with the guest OS. Sometimes applications have a hard time running in a different version of the same OS the application was developed for. This is also very inefficient and time-consuming.

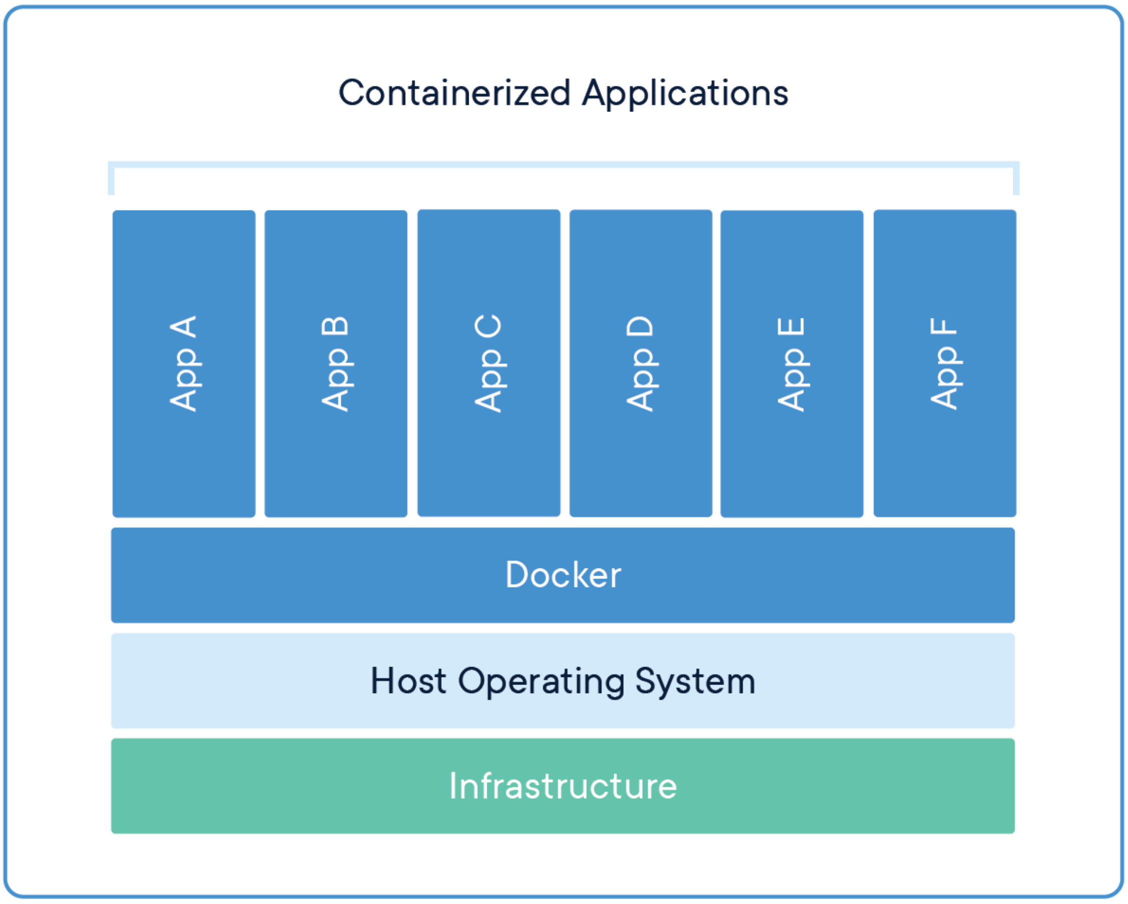
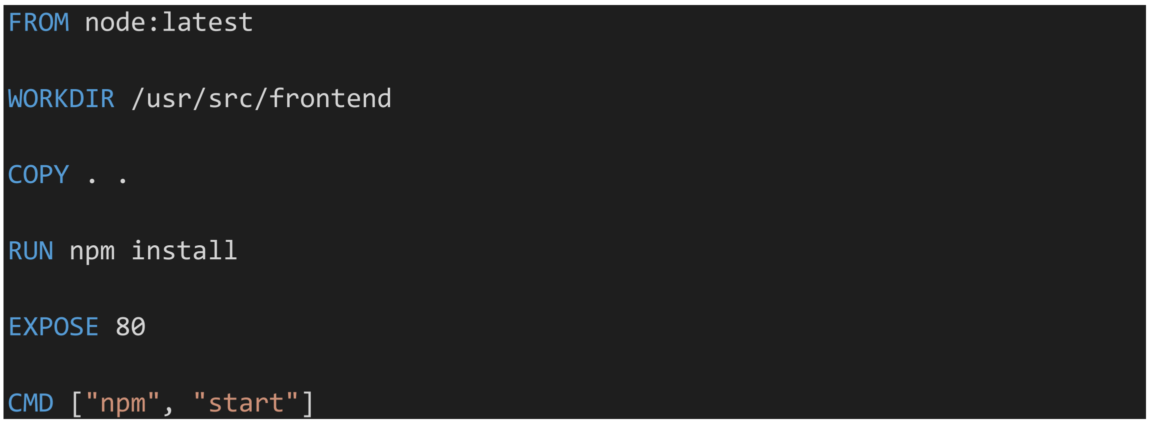


Figure 10. Container architecture. image copied from docker website [6].

Docker containers share the host operating system (see figure 10), which means any machine that runs Docker can run the application, making containers extremely portable and deployable fast. Creating a Docker container is done by defining a Dockerfile.

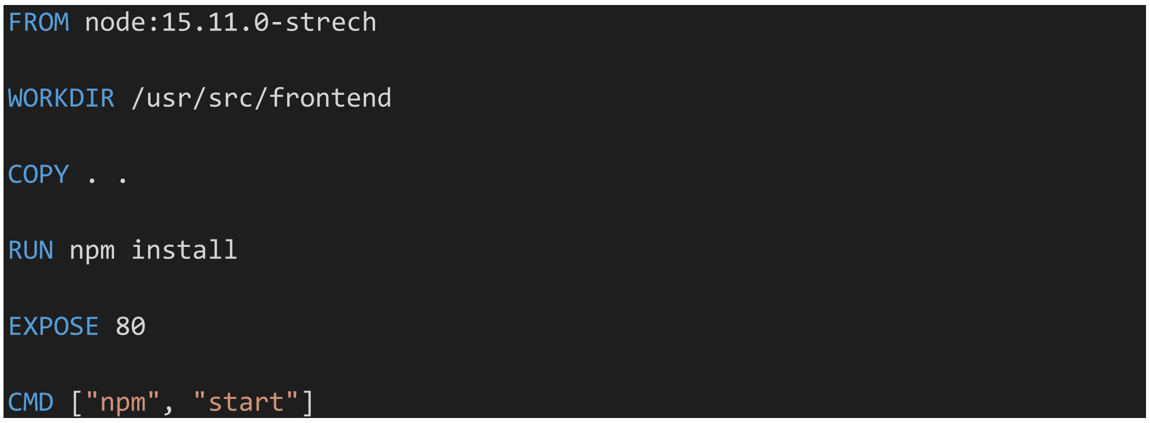
In a Dockerfile, the instructions are defined to build the container image. Each instruction is a layer, and when an instruction is changed, only the layer is changed, enabling fast and light updating of the container [6.]



```
FROM node:latest

WORKDIR /usr/src/frontend

COPY . .

RUN npm install

EXPOSE 80

CMD ["npm", "start"]
```

Figure 11. Example Dockerfile

Images are often made from other images, just like in the example, figure 11. The example Dockerfile creates a container that runs a frontend application on port 80 from the node base image. Containers are usually stored in an online registry publicly or privately, like Docker Hub and JFrog Artifactory.



```
FROM node:15.11.0-strech

WORKDIR /usr/src/frontend

COPY . .

RUN npm install

EXPOSE 80

CMD ["npm", "start"]
```

Figure 12. Dockerfile with version defined

Private image registries are considered safer because unauthorized parties cannot mess with the container images. Unofficial images might contain malicious content that is not wished for. The image user should also be aware of the version because if a malicious image is pushed to the registry, those who have the “latest” version in use might get harmed (see figure 12). It also makes it harder to determine which version was used and which version should be rollbacked in case of everything breaking down.

2.5 Infrastructure as code

Managing IT infrastructure nowadays is not as painful as it used to be. Creating cloud infrastructure can be made using different IaC solutions and tools. Instead of manually clicking from the web interface of the cloud provider, provisioning can be done through structured files. IaC does not limit computers and virtual machines and can spin up other resources that the provider might have, like service accounts.

Managing the infrastructure as code has many benefits. Users can ensure consistency between two similar environments. In addition, one of the main factors why there is a significant move towards managing IaC is speed. Once the user has templated the infrastructure, they can easily replicate it and modify it to the user's needs, which also means it can be scaled easily [7.]

The tools used to create infrastructure as code do not vary much since major companies heavily dominate the market. The three big cloud providers have their services to define infrastructure as code, such as AWS CloudFormation, Google Cloud Resource Manager, and Azure Resource Manager. There are also third-party tools made like Hashicorp's Terraform, which currently is the market leader.

It is important to create good quality infrastructure code to minimize the chance of security breaches. There are some static analysis tools developed for this purpose. They were used in making the project. Hard-coded secrets in code or credentials are one of the examples faced in the real world. [8] Other issues might be suspicious comments, usage of HTTP without TLS, or default users with administrator privileges.

Terraform will be focused in this thesis since it was used to make the infrastructure of the cluster.

2.6 Terraform

Terraform uses a variation of HCL to define infrastructure resources. HCL stands for Hashicorp Configuration Language and is used mainly by Hashicorp products. HCL uses blocks to define resources. [9]

```
resource "google_container_cluster" "gke-cluster" {
  name                     = "example-cluster"
  location                 = "europe-north1-a"
  initial_node_count       = 1
  remove_default_node_pool = true

  master_auth {
    username = ""
    password = ""

    client_certificate_config {
      issue_client_certificate = false
    }
}
```

Figure 13. Example resource block

As can be seen from figure 13, blocks have types like the resource. After the block there are two labes that are expected by the resource block, the resource type, and the resource name. After labels the body of the block is defined, where more blocks and arguments are provided that might required by the resource.

```
provider "google" {
  project = "example-project"
  region  = "europe-north1"
}
```

Figure 14. Example provider block

Before creating certain resources, provider needs to be defined (see figure 14). Providers are plugins made for Terraform to enable interaction with for example GCP. The providers add the resources and data sources to Terraform. There are different kinds of providers, official, verified, community-made and archived ones. Official providers are owned and maintained by Hashicorp, and verified providers are made by partnering companies [10.]

```
module "ClusterModule" {
  source   = "/path/to/module"

  name     = "Example cluster"
  location = "europe-north1"
}
```

Figure 15. Example module block

It is a best practice to package and reuse configurations when possible, which can be achieved in Terraform via modules. The modules are containers of multiple resource definitions in a collection of files. Files are stored in the same directory, and the source of the module blocks is pointed at the directory to tell Terraform where they exist (see figure 15).

```
variable "name" {
  type        = string
  default     = "example-name"
  description = "Name of the example resource"
}

variable "location" {
  type        = string
  default     = "europe-north1"
  description = "Location of the example resource"
}

output "gateway_ip" {
  value       = google_compute_network.vpc_network.gateway_ipv4
  description = "Example network gateway ip"
}

output "network" {
  value       = google_compute_network.vpc_network.self_link
  description = "Example network output"
}
```

Figure 16. Example variable and output blocks

The users can pass the needed arguments from modules to resources via variables and access other resource attributes via outputs. The outputs can be viewed as the return values of Terraform. The variables help customize modules without changing the source code. The variables and outputs need to be defined in their blocks (see figure 16).

3 Implementation

The starting of the implementation was started by creating accounts and providing the needed permissions for the GCP project. The work was started immediately. Before the project actually started, the resources GCP provides were looked at, and cost efficient calculations were made for the infrastructure.

3.1 Creating the network

First a Virtual Private Cloud Network, aka VPC was created. Two separate networks were created. First for the production environment and second for the development and test environment. Internal addresses and ingress traffic do not cost, but external addresses and egress traffic do. However, some services that handle the ingress traffic do cost like Load Balancers, Cloud NAT, and Protocol forwarding [11.]

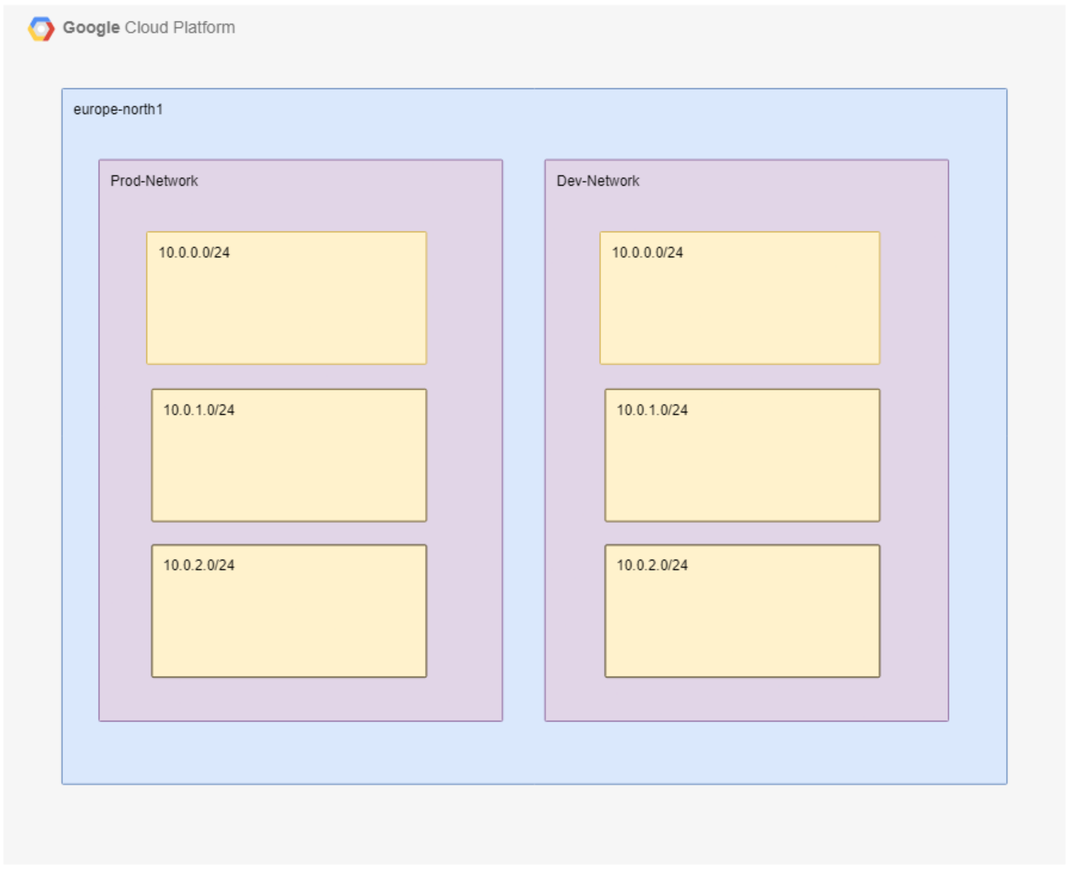


Figure 17. Network architecture

A private Kubernetes cluster needs subnet ranges defined for pods and services, so they were defined as shown in figure 17. the development network was separated from the production network because they did not need to communicate. Private cluster nodes do not have external IP addresses, so they cannot access the internet. The application running in the cluster needed to communicate with external services, so enabling of outgoing traffic needed to be done.

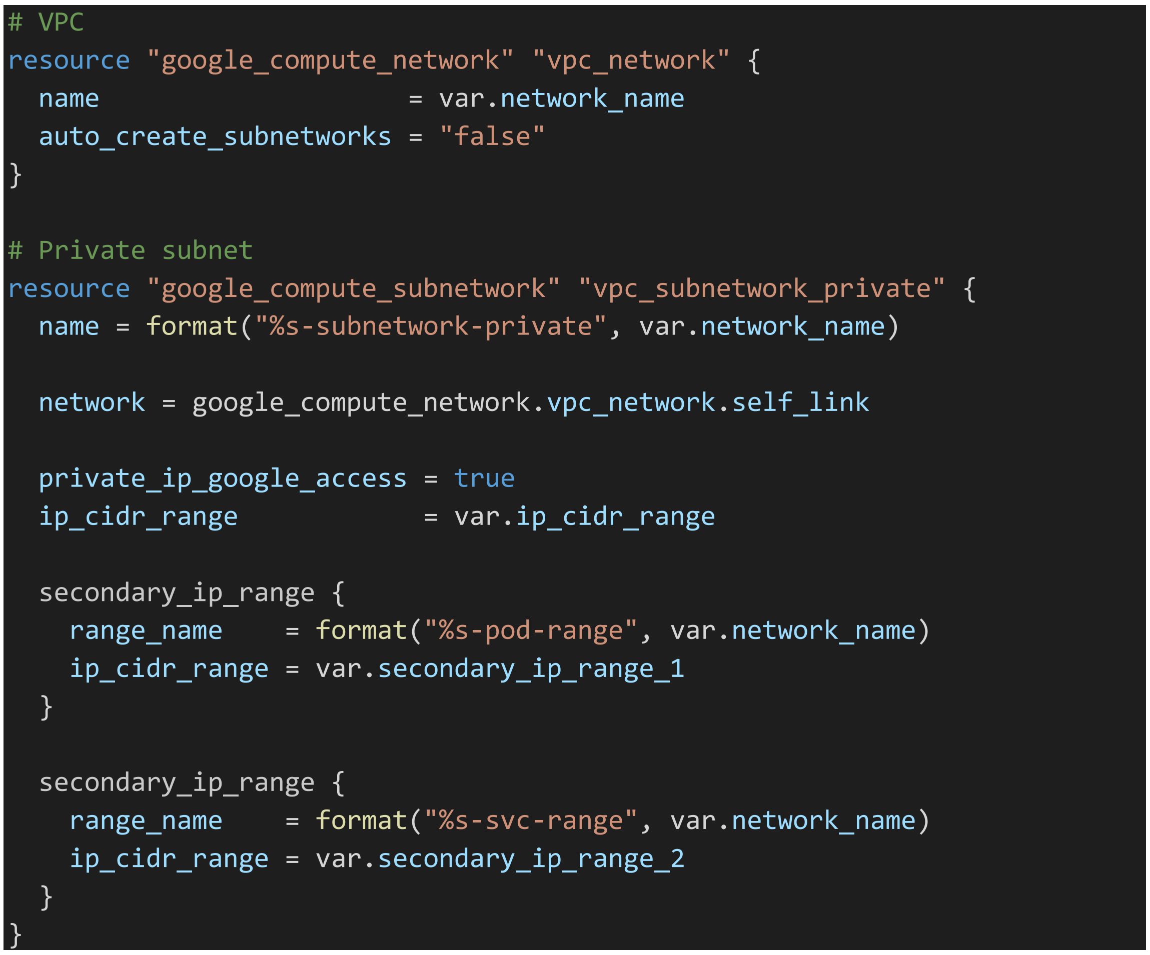

```
# VPC
resource "google_compute_network" "vpc_network" {
  name                    = var.network_name
  auto_create_subnetworks = "false"
}

# Private subnet
resource "google_compute_subnetwork" "vpc_subnetwork_private" {
  name = format("%s-subnetwork-private", var.network_name)

  network = google_compute_network.vpc_network.self_link

  private_ip_google_access = true
  ip_cidr_range            = var.ip_cidr_range

  secondary_ip_range {
    range_name    = format("%s-pod-range", var.network_name)
    ip_cidr_range = var.secondary_ip_range_1
  }

  secondary_ip_range {
    range_name    = format("%s-svc-range", var.network_name)
    ip_cidr_range = var.secondary_ip_range_2
  }
}
```

Figure 18. Terraform code for the VPC network and subnet with secondary ranges.

Allowing internal traffic between the applications and Google API's were done as seen in figure 18.This helped to access google container registry easily.

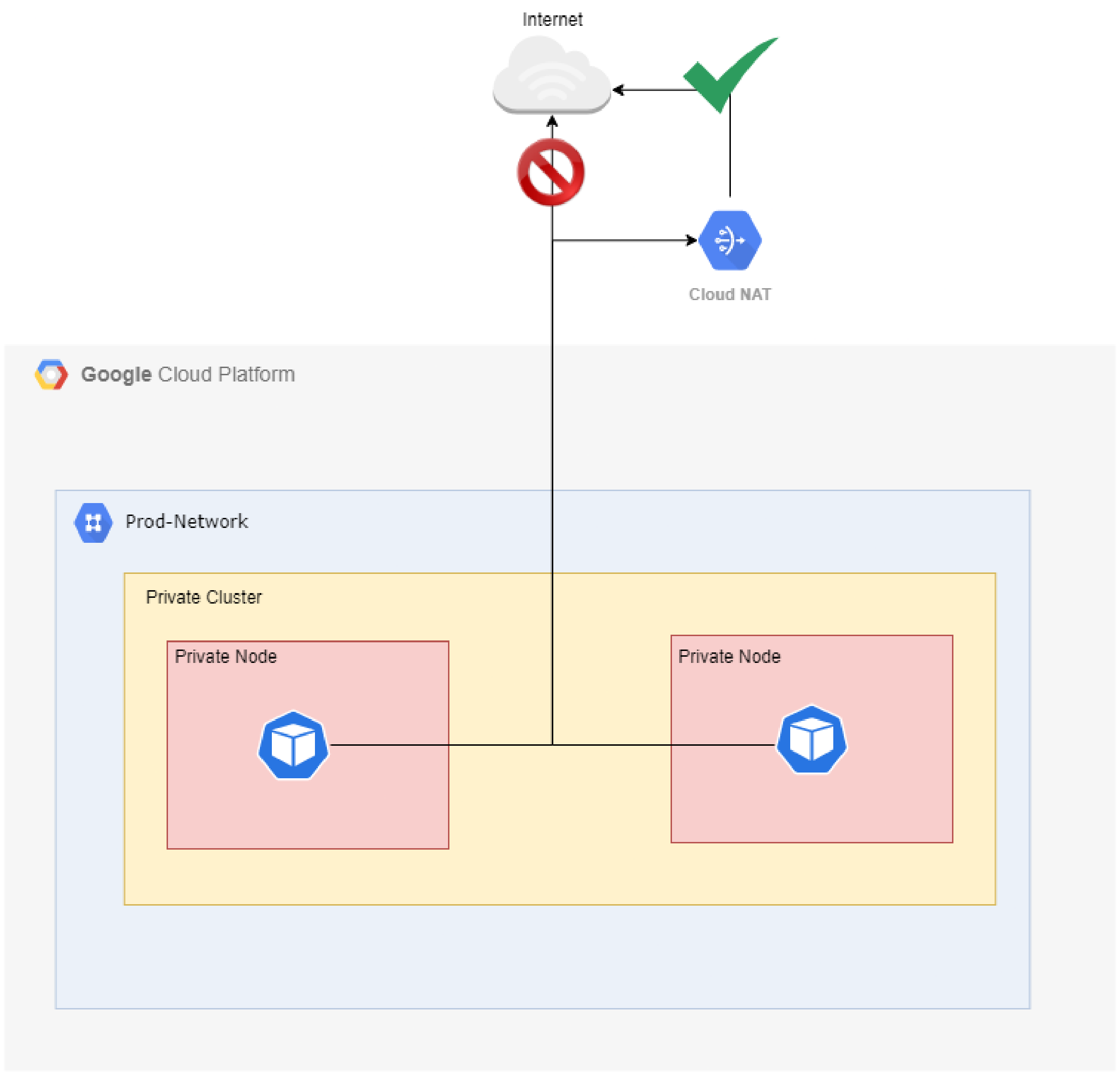
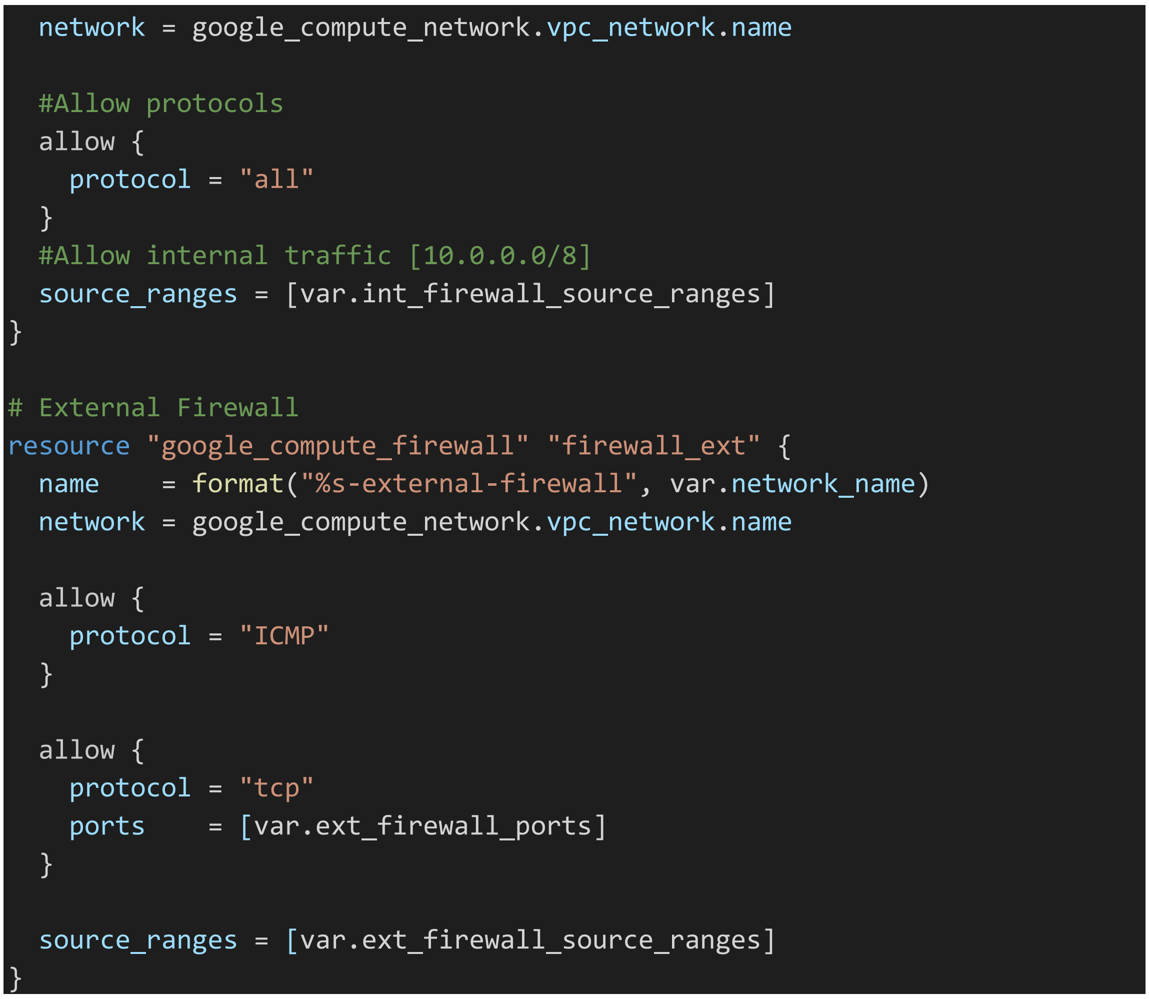


Figure 19. Network architecture with NAT. based on information from Google documentation [12].

Enabling internet connectivity to private nodes is done with the cloud router and the cloud NAT. NAT provides a public IP to the nodes, which they can use to access the internet. (See figure 19.) Cloud Router is required by Cloud NAT and is used to route traffic from private instances like the ones that were used in this project.

After allowing internet access from the private cluster, traffic between the internal services and application needed to be allowed. Google Cloud Platform offers firewall rules that can be applied to networks. ICMP, SSH and HTTPS traffic was allowed from outside the VPC. All traffic between the internal components were allowed [12.]

```
# Internal firewall
resource "google_compute_firewall" "firewall_int" {
  name    = format("%s-internal-firewall", var.network_name)
```



```
  network = google_compute_network.vpc_network.name

  #Allow protocols
  allow {
    protocol = "all"
  }
  #Allow internal traffic [10.0.0.0/8]
  source_ranges = [var.int_firewall_source_ranges]
}

# External Firewall
resource "google_compute_firewall" "firewall_ext" {
  name    = format("%s-external-firewall", var.network_name)
  network = google_compute_network.vpc_network.name

  allow {
    protocol = "ICMP"
  }

  allow {
    protocol = "tcp"
    ports    = [var.ext_firewall_ports]
  }

  source_ranges = [var.ext_firewall_source_ranges]
}
```

Figure 20. Firewall rules in the Terraform configuration.

The terraform configuration in figure 20 is missing one piece that was made in the rules. SSH traffic was allowed only from a VPN and SSH key holders which helped in cutting unauthorized access to the platform. With the configurations demonstrated in figures 18 and 20,

Networking played a crucial part in securing the environment within the budget since most of the networking features are low cost or free of charge. In the next chapter a look at the Kubenetes engine and cost optimization will be done.

3.2 Creating the Kubernetes cluster

Before creating the Kubernetes cluster, a closer look at the options to optimize the cost needed to be made. Optimizing of the costs were done by adjusting the following specifications:

- Machine type
- Size
- Region
- Committed use discounts
- Autoscaling

For the machine type, the n1-standard-2 was chosen since it was the closest to the old setup, which had proved to be working without any problems. Ramping down of the servers was done for the development environment since it did not handle as much traffic as the production.

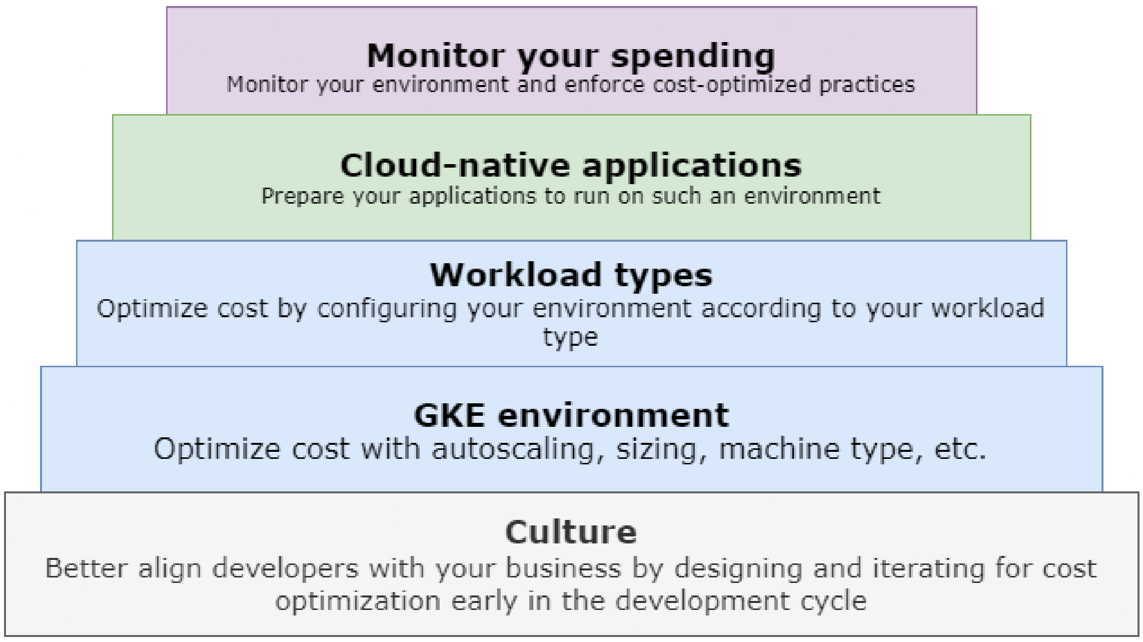
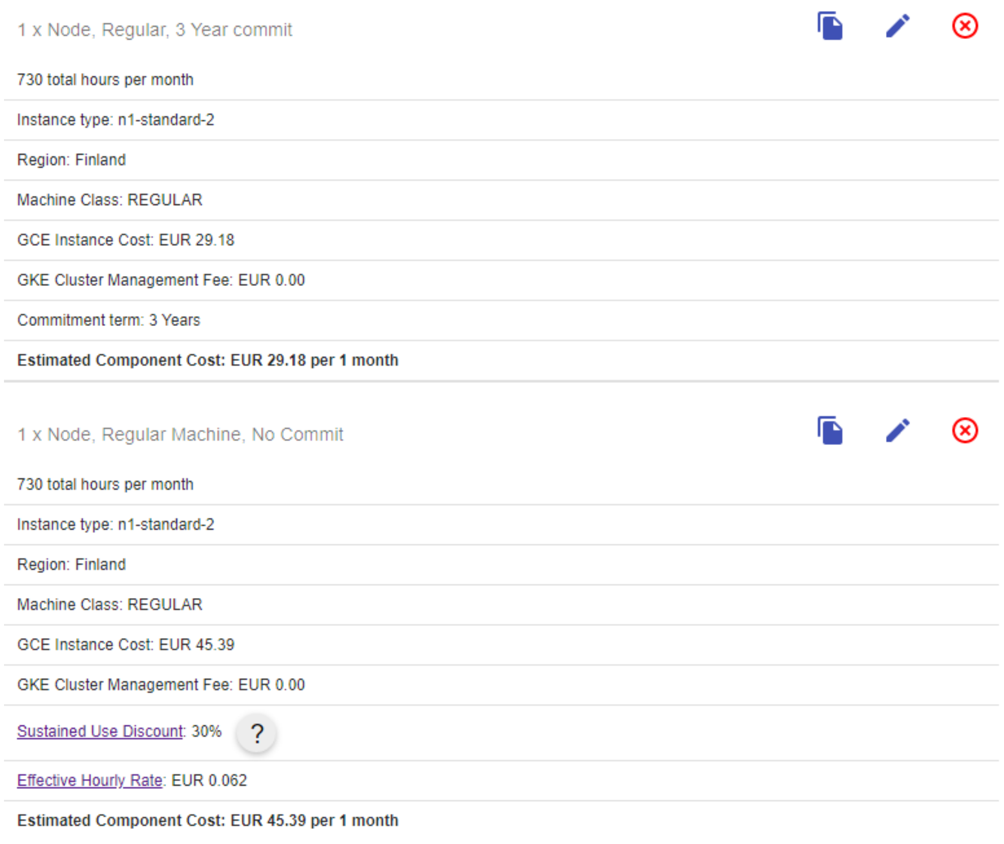

Monitor your spending
Monitor your environment and enforce cost-optimized practices
Cloud-native applications
Prepare your applications to run on such an environment
Workload types
Optimize cost by configuring your environment according to your workload type
GKE environment
Optimize cost with autoscaling, sizing, machine type, etc.
Culture
Better align developers with your business by designing and iterating for cost optimization early in the development cycle

Figure 21. GKE best practice diagram

The auto-scaling and auto-repair functions for the development environment were also disabled for the same reasoning. The best practice diagram in figure 21 was followed to achieve the best results.



1 x Node, Regular, 3 Year commit

730 total hours per month

Instance type: n1-standard-2

Region: Finland

Machine Class: REGULAR

GCE Instance Cost: EUR 29.18

GKE Cluster Management Fee: EUR 0.00

Commitment term: 3 Years

Estimated Component Cost: EUR 29.18 per 1 month

1 x Node, Regular Machine, No Commit

730 total hours per month

Instance type: n1-standard-2

Region: Finland

Machine Class: REGULAR

GCE Instance Cost: EUR 45.39

GKE Cluster Management Fee: EUR 0.00

Sustained Use Discount: 30%

Effective Hourly Rate: EUR 0.062

Estimated Component Cost: EUR 45.39 per 1 month

Figure 22. Screenshot of the Google cloud price calculator

Some testing was done using Google's price calculator. First, the customer was contacted and asked if they wanted to commit to using Google Cloud for 1 or 3 years because, this could reduce the cost by about 33% according to the calculator. The estimated cost of the three-year committed one node cluster is ~€30/month and without the same setup is ~€45/month (see figure 22). Preemptible nodes are the cheapest option, but they are not reliable if uptime is essential. Google might free these resources for other tasks if needed [13.]

The region of the cluster needed to be in Finland since it was considered important to keep data in finland. There would have been cheaper regions to host the cluster in, for example, us-west1, but it was necessary to think about the possible problems, such as higher response times that come with having data across the globe in relation to the users.

After signing up for the committed use discount, provisioning of the cluster was started. Securing the GKE control plane was done using authorized network functionality built into GKE, which allowed controlling the actors who could connect to the Kubernetes control plane. Another option was to make a private endpoint that could be accessed using a bastion host, but that did not fit this setup.

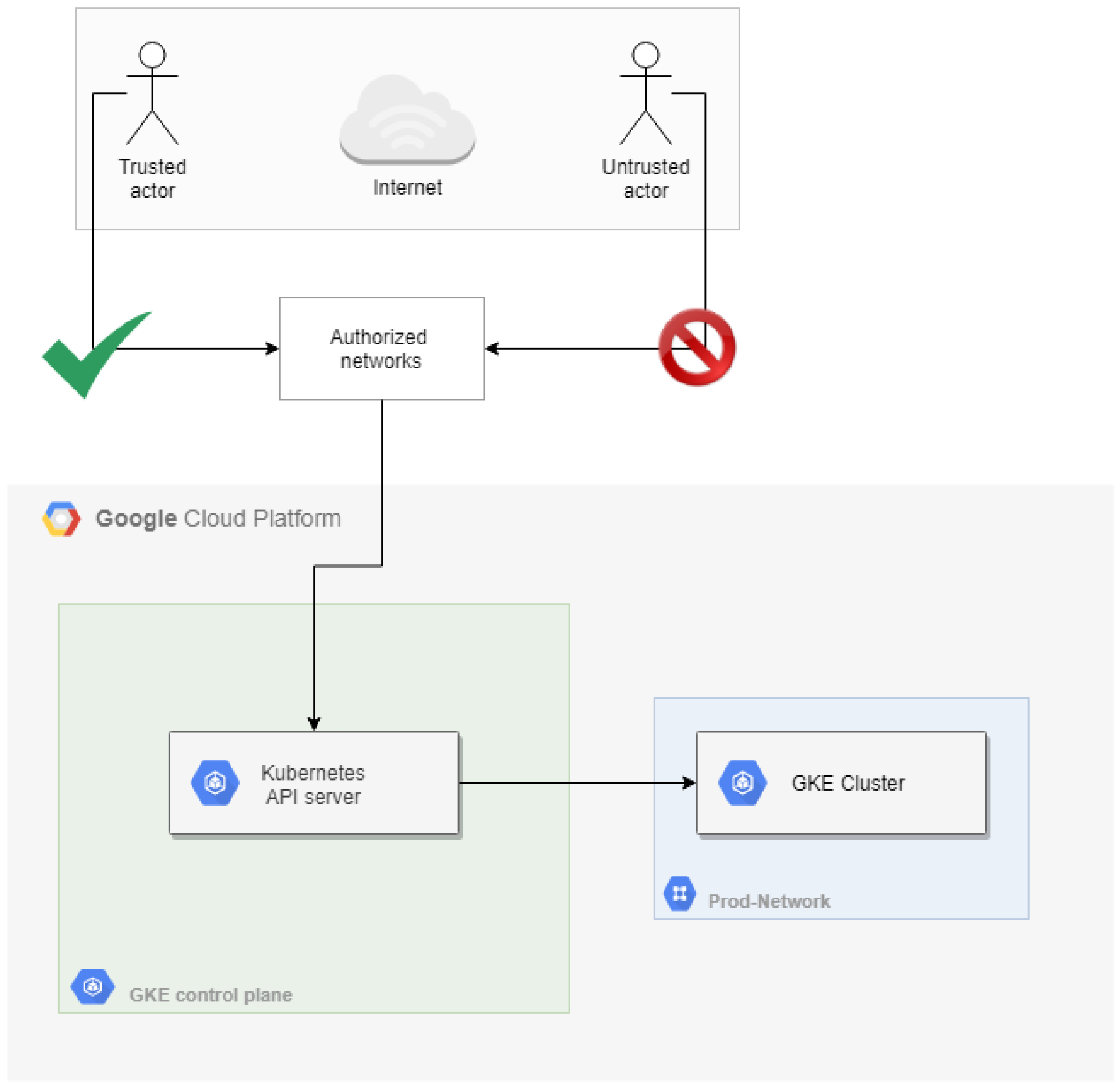
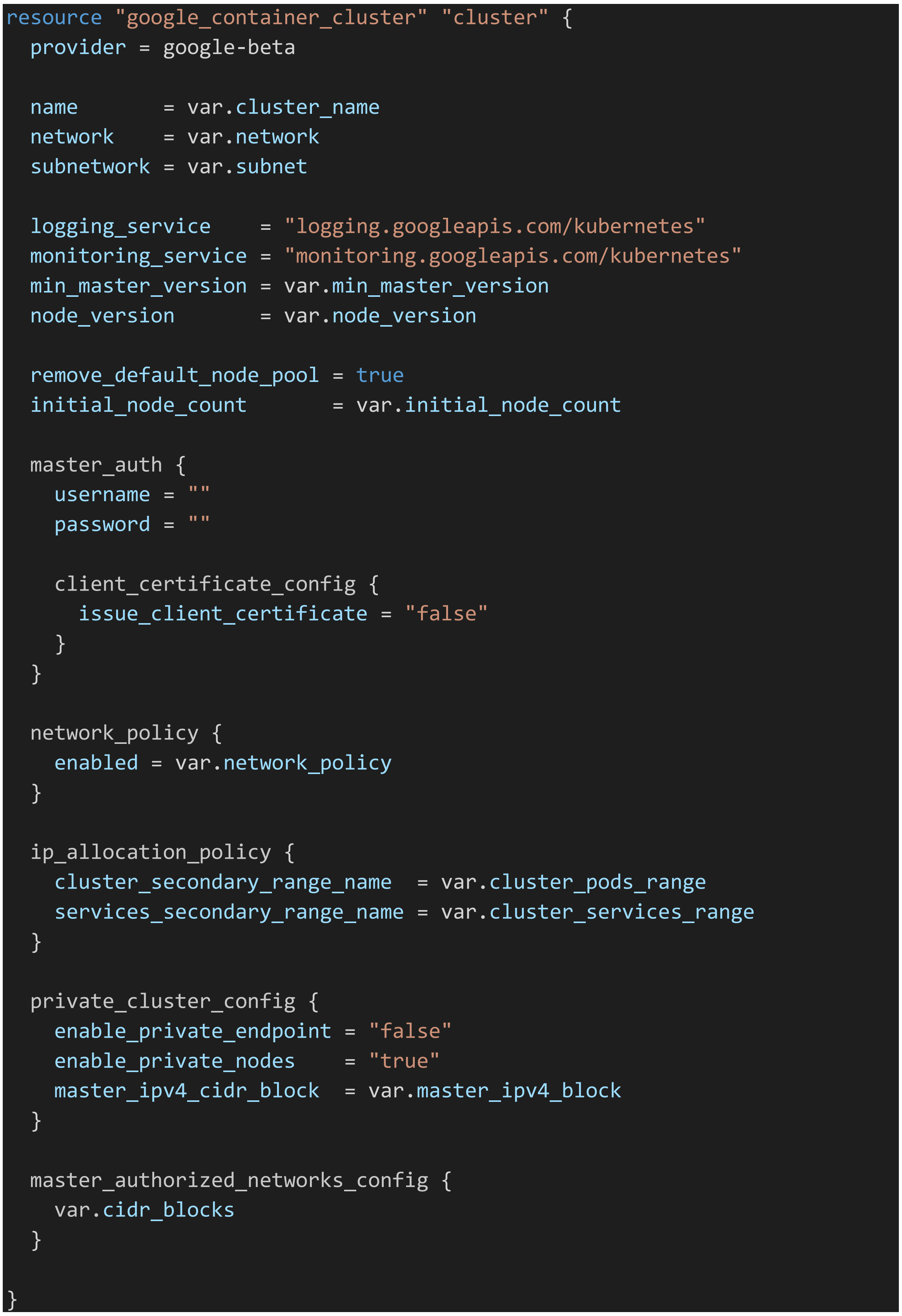


Figure 23. Diagram of authorized networks

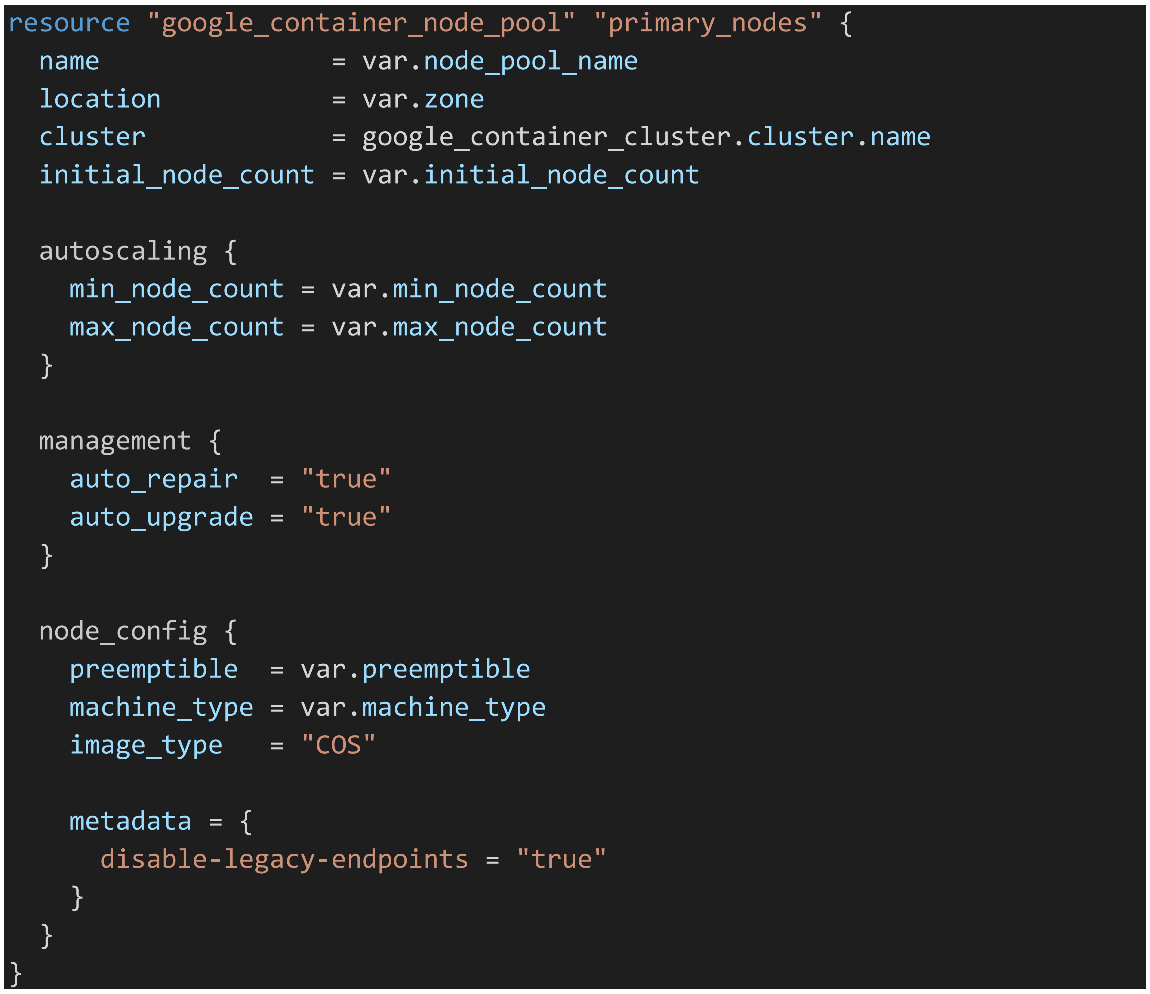
traffic to the control plane was allowed from the company network and the VPN. This way, unwanted actors could be limited from getting hand on to the control plane. (See figure 23.)



```
resource "google_container_cluster" "cluster" {
  provider = google-beta

  name       = var.cluster_name
  network    = var.network
  subnetwork = var.subnet

  logging_service    = "logging.googleapis.com/kubernetes"
  monitoring_service = "monitoring.googleapis.com/kubernetes"
  min_master_version = var.min_master_version
  node_version       = var.node_version

  remove_default_node_pool = true
  initial_node_count       = var.initial_node_count

  master_auth {
    username = ""
    password = ""

    client_certificate_config {
      issue_client_certificate = "false"
    }
  }

  network_policy {
    enabled = var.network_policy
  }

  ip_allocation_policy {
    cluster_secondary_range_name  = var.cluster_pods_range
    services_secondary_range_name = var.cluster_services_range
  }

  private_cluster_config {
    enable_private_endpoint = "false"
    enable_private_nodes    = "true"
    master_ipv4_cidr_block  = var.master_ipv4_block
  }

  master_authorized_networks_config {
    var.cidr_blocks
  }

}
```

Figure 24. Terraform code for Google Kubernetes Cluster

After having the cluster defined and configured (see figure 24), node pool needed to be defined for the cluster. As already mentioned, the n1-standard-2 was chosen as the machine type for the nodes. The maximum size of the pool was 2 for the production and 1 for the development environment. Autoscaling will handle spinning up new nodes in case of more traffic.



```
resource "google_container_node_pool" "primary_nodes" {
  name               = var.node_pool_name
  location           = var.zone
  cluster            = google_container_cluster.cluster.name
  initial_node_count = var.initial_node_count

  autoscaling {
    min_node_count = var.min_node_count
    max_node_count = var.max_node_count
  }

  management {
    auto_repair  = "true"
    auto_upgrade = "true"
  }

  node_config {
    preemptible  = var.preemptible
    machine_type = var.machine_type
    image_type   = "COS"

    metadata = {
      disable-legacy-endpoints = "true"
    }
  }
}
```

Figure 25. GKE Node pool as Terraform code.

auto-upgrade and repair functionalities was also enabled to keep nodes healthy, running, and up to date always without a difficulty (see figure 25). This also brings more security because new binaries might contain security fixes and auto-update takes care of them almost instantly.

The cost of the clusters came to be about €120/monthly but with the committed use the discount ended up being about €50/monthly which was a huge save.

3.3 Creating the database

Using CloudSQL was considered better approach than hosting a database instance inside the cluster. This offered automated backups, high availability, and failover, with the trade of it slashing the budget a little. The CloudSQL pricing varies a lot based on the following factors.

- Storage provisioned
- CPU count for the instance
- Amount of memory on the instance
- Data location
- Network traffic volume
- Number of IP addresses

A private IP for the database was only needed, so there were no additional costs from external IP allocation. The database did not need to handle much traffic, so only one virtual CPU was enough with 4 GB of RAM. Storage was set to 15GB of SSD with automated storage increase in case of the storage filling up even though it was considered highly unlikely.

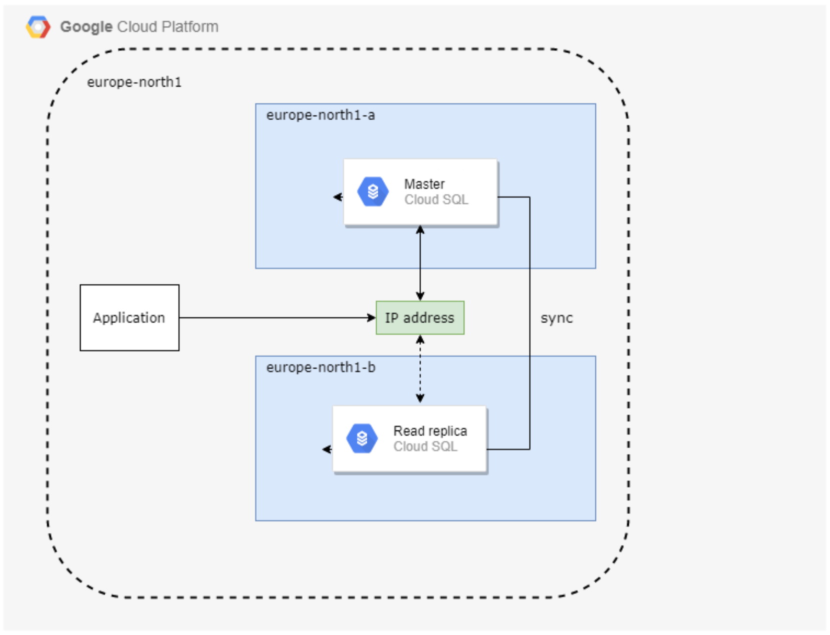
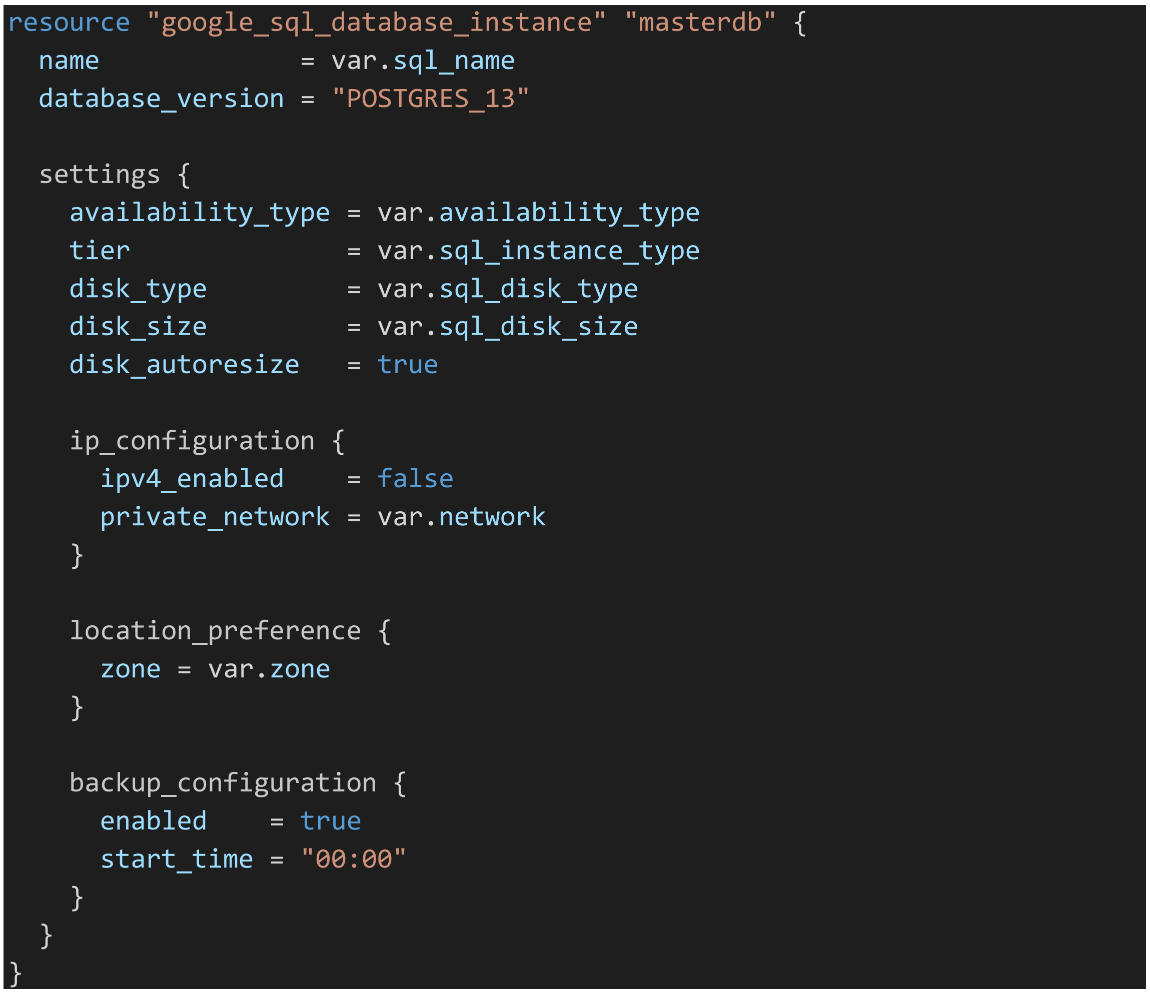


Figure 26. CloudSQL high availability diagram

High availability was achieved by placing a standby instance in another zone in the same region (see figure 26).



```
resource "google_sql_database_instance" "masterdb" {
  name             = var.sql_name
  database_version = "POSTGRES_13"

  settings {
    availability_type = var.availability_type
    tier              = var.sql_instance_type
    disk_type         = var.sql_disk_type
    disk_size         = var.sql_disk_size
    disk_autoresize   = true

    ip_configuration {
      ipv4_enabled    = false
      private_network = var.network
    }

    location_preference {
      zone = var.zone
    }

    backup_configuration {
      enabled    = true
      start_time = "00:00"
    }
  }
}
```

Figure 27. CloudSQL terraform code.

auto backups were enabled and timed to be at midnight, as shown in figure 27. This way, data loss could be minimized in case of disastrous scenarios where the database would fail. Security comes with a private IP address which can only be accessed inside the network. Cloud SQL also provides an option to control access to databases on an instance level, which helps minimize unwanted actors from accessing the database.

The whole Cloud SQL setup cost is a slight hit on the budget, costing ~€60 monthly. Being fully managed by Google guarantees less maintenance cost; automated backups ensure business continuity. Security updates and compliance is under Google's territory as well. This secondhandedly compensates the costs that might come from self-hosting the database.

4 Discussion

As already mentioned, fast migration towards containerized environments is prone to security-related issues (see Chapter 2.4). Container runtime like Docker might be vulnerable which mitigates the security of the whole cluster. Other community members have been trying to tackle the issue by developing safer container run times. A study by a Swedish group, however, suggests that it might lead even to a trade five times higher in performance [14.]

Secure storing of secrets and credentials is also one of the biggest issues that might lead to problems. Some criminals might use the credentials for their purpose, for example, cryptojacking. Cryptojacking is malware that enables cryptocurrency mining on the infected computer. In 2018 a group of hackers was able to access Tesla's unsecured Kubernetes dashboard and get Cloud credentials from there [15.] The car manufacturer suffered losses and suffered a data breach. This issue was tackled by blocking untrusted actors from accessing the administrator panel.

Maximizing security, minimizing cost, and meeting customer demand have always been the impossible equation that still to this day needs work. New technology is rapidly changing and the maximum potential of the state of the art technology might never be reached in such a short time. Maximizing the cost efficiency of Kubernetes is different for each application and needs testing to avoid major drawbacks that might come from resource allocation [16.]

Some studies suggest placing containers into existing resources or scheduling shutdowns for resources that might not be needed at certain times. Most of the publications have one suggestion in common which is auto-scaling. For example, a study by Z. Zhong and R. Buyya demonstrates that the mentioned strategies might cut down costs even 30% [17.] Auto-scaling is described as one of the most efficient ways to cut costs in cloud computing. The committed use of reserved instances was found to be almost as effective in this thesis.

Putting a price on security is hard and often analyzed by many institutions and companies. Case studies suggest that having all the security is not realistic and that the question of how much is enough is hard to answer [18.] Bryan Paynes' paper suggests that securing a cloud application is high due to poor foundations for it. In this study however, it was proven that costs do not increase highly and the foundations might have come a bit further.

Crimes have moved more towards the cyber world with the rapid adoption of new computing technology with no end in sight [19.] It is expected to grow rapidly which makes security in this field critical and important. That is one of the reasons why the cost of security should be low and easily available for all users.

5 Conclusions

The goals set before the work process were met by optimizing costs using the best practices. These included autoscaling, machine-type, size, and zone. The whole project ended up costing under €250/month, which can be considered reasonable. The security of the cluster was on the required level and and it is safe to put the cluster into production. Security was hardened by restricting access to the administrator panel of Kubernetes and by allowing only traffic from known sources to access critical infrastructure.

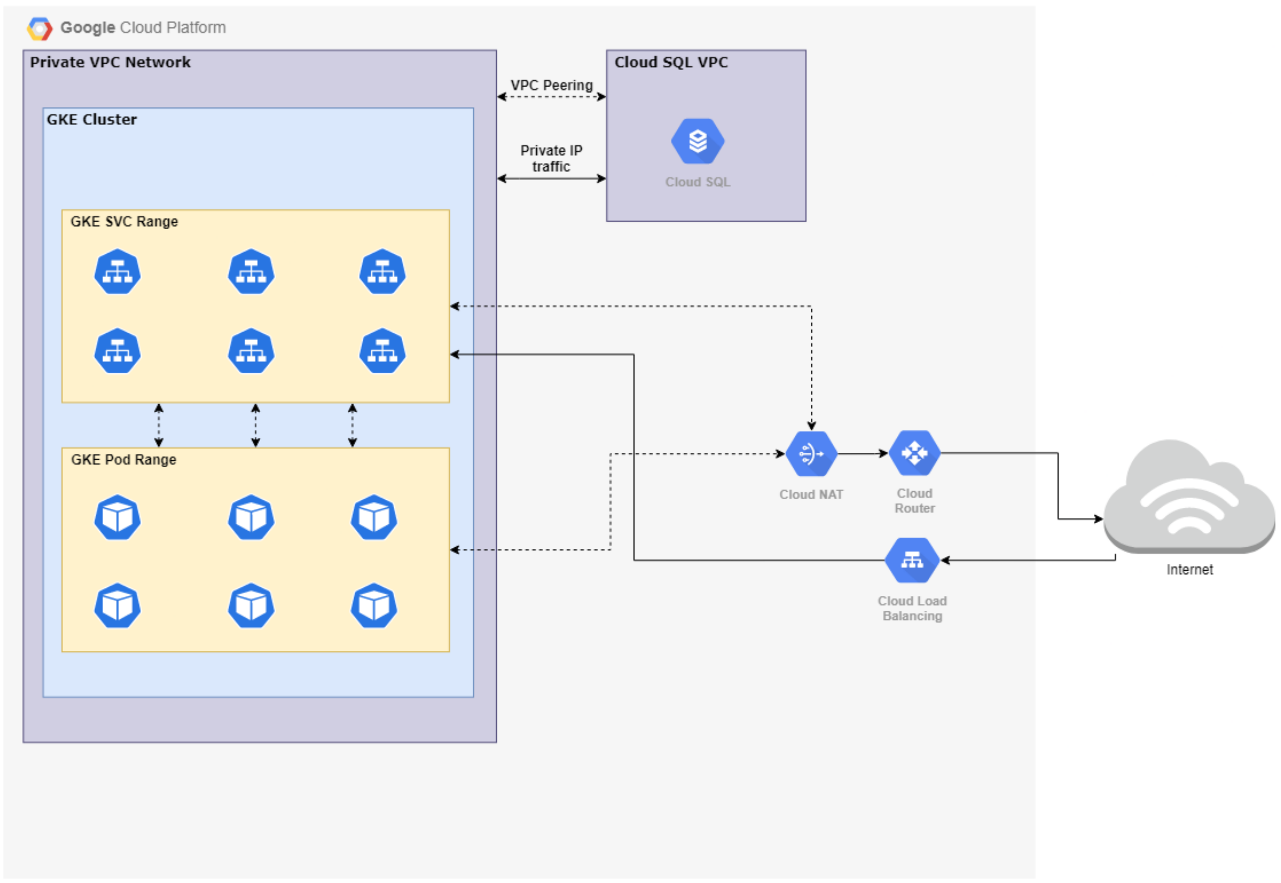


Figure 28. Solution diagram

The project outcome is illustrated in figure 28. The goals set in the beginning were met and the original research question was answered. The project showed that there is no need to thoughtlessly pour money into features to secure a Kubernetes cluster in the cloud. However, they bring an extra layer of security or help if the user has no prior knowledge of networking. Following the guides made by the cloud providers can take users far in securing the cloud infrastructure. Carrying out this project required motivation, experience in networking, and communication skills with the customer. Adhering to to the work schedule and budget also required some attention.

Regarding security, there are usually two different aspects to cloud and security. The first view usually considers the cloud environments unsafe for many reasons: bigger attack surface and lack of visibility over the infrastructure layer. The second group usually considers the cloud environments safe because it is kept by companies that have large budgets to invest into security. Like most of the time, the truth usually lies somewhere in between the two the groups.

The infrastructure managed to stay easily under the €300/month mark, which was given as the ideal budget. Costs could have gone even more under the budget by hosting the database inside the cluster, but the trade made sense in this case.

Future improvements for this project will be perfecting the auto-scaling configuration, utilization of new GKE functionalities, improving the request rate in Cloud SQL, and hardening the network security if needed.

The objectives set in the beginning of the project were met, and migration was considered successful.

References

1 Google Authors. (2021). What is CloudSQL? Available at: https://cloud.google.com/sql/docs/introduction (Accessed 20.2.2021)

2 Gwan-Hwan Hwang and Shih-Kai Fu. (2016). Proof of violation for trust and accountability of cloud database systems. Available at https://doi.org/10.1109/CCGrid.2016.27 (Accessed 26.4)

3 Kubernetes Authors. (2021). What is Kubernetes? Available at: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/ (Accessed 20.2.2021)

4 Kubernetes Authors. (2021). Services. Available at: https://kubernetes.io/docs/concepts/services-networking/service/ (Accessed 20.2.2021)

5 Sandeep Dinesh. (2018). Kubernetes NodePort vs LoadBalancer vs Ingress? When should I use what? Available at https://medium.com/google-cloud/kubernetes-nodeport-vs-loadbalancer-vs-ingress-when-should-i-use-what-922f010849e0 (Accessed 20.2.2021)

6 Docker Authors. (2021). Docker overview. Available at: https://docs.docker.com/get-started/overview/ (Accessed 14.3.2021)

7 Amazon Authors. (2021). Infrastructure as code. Available at: https://docs.aws.amazon.com/whitepapers/latest/introduction-devops-aws/infrastructure-as-code.html (Accessed 27.3.2021)

8 Akond Rahman, Chris Parnin, and Laurie Williams. (2019). The seven sins: security smells in infrastructure as code scripts. Available at https://doi.org/10.1109/ICSE.2019.00033 (Accessed 26.4)

9 Terraform Authors. (2021). Introduction to Terraform. Available at: https://www.terraform.io/intro/index.html (Accessed 27.3)

10 Terraform Authors. (2021). Providers. Available at: https://www.terraform.io/docs/language/providers/index.html (Accessed 27.3)

11 Google Authors. (2021). VPC pricing. Available at: https://cloud.google.com/vpc/pricing (Accessed 2.4)

12 Google Authors. (2021). Creating a private cluster. Available at: https://cloud.google.com/kubernetes-engine/docs/how-to/private-clusters (Accessed 13.4)

13 Google Authors. (2021). Best practices for running cost-optimized Kubernetes applications on GKE. Available at https://cloud.google.com/solutions/best-practices-for-running-cost-effective-kubernetes-applications-on-gke (Accessed 13.4)

14 W. Viktorsson, C. Klein and J. Tordsson. (2021). Security-Performance Trade-offs of Kubernetes Container Runtimes. Available at https://ieeexplore.ieee.org/document/9285946 (Accessed 26.4)

15 D. Goodin. (2018). Tesla cloud resources are hacked to run cryptocurrency-mining malware. Available at https://arstechnica.com/information-technology/2018/02/tesla-cloud-resources-are-hacked-to-run-cryptocurrency-mining-malware/ (Accessed 26.4)

16 Stef Verreydt, Emad Heydari Beni, Eddy Truyen, Bert Lagaisse, and Wouter Joosen. (2019). Leveraging Kubernetes for adaptive and cost-efficient resource management. Available at https://doi.org/10.1145/3366615.3368357 (Accessed 26.4)

17 Zhiheng Zhong and Rajkumar Buyya. (2020). A Cost-Efficient Container Orchestration Strategy in Kubernetes Based Cloud Computing Infrastructures with Heterogeneous Resources. Available at https://doi.org/10.1145/3378447 (Accessed 26.4)

18 Bryan D. Payne. (2014). Reducing the Cost of Security in the Cloud. Available at https://doi.org/10.1145/2664168.2664184 (Accessed 26.4)

19 Vijay Varadharajan. (2011). Rethinking cyber security. Available at https://doi.org/10.1145/2070425.2070428 (Accessed 26.4)