Use Case: Deploying a web application with Scaleway Kubernetes Kapsule using a full isolation pool and Public Gateway

Overview

A fictive mid-sized e-commerce company, ShopSecure, wants to deploy a WooCommerce-based web application on Scaleway's Kubernetes Kapsule. The application requires high security, scalability, and controlled external access. To achieve this, ShopSecure opts for a **full isolation pool** to ensure that worker nodes have only private IP addresses, communicating within a Private Network, and a **Public Gateway** to manage secure external access to the WooCommerce application. This setup ensures that internal cluster traffic is isolated from the public internet, while external users can access the application through a controlled entry point.

Objectives

- **Security**: Isolate all internal cluster communication within a private network to prevent exposure to the public internet.
- **Scalability**: Enable auto-scaling to handle fluctuating traffic, such as during sales events.
- **Controlled access**: Use a public gateway to manage and secure external traffic to the WooCommerce application.
- **Resilience**: Ensure high availability by deploying across multiple availability zones (AZs).

Architecture

The architecture involves:

- **Kubernetes Kapsule cluster**: A managed Kubernetes cluster hosted in Scaleway's Paris region (fr-par).
- **Full isolation pool:** Worker nodes configured with only private IP addresses, communicating within a Scaleway Private Network.

- **Public Gateway**: A Scaleway Public Gateway with Dynamic NAT enabled to route outgoing traffic from the private network to the public internet and allow controlled inbound access.
- **Load Balancer**: A Scaleway Load Balancer to distribute external traffic to the WooCommerce application's pods.
- **Containerized WooCommerce application:** A WooCommerce application (based on WordPress) deployed as pods, accessible externally via an Ingress controller, with a MySQL database for persistence.

Prerequisites

- A **Scaleway account** with IAM permissions to create and manage Kubernetes clusters, VPC, Private Networks, and Public Gateways.
- Created a VPC in the desired region to deploy
- Scaleway CLI or console access for configuration.
- A containerized **WooCommerce application image** (WordPress with WooCommerce plugin) stored in Scaleway Container Registry or another registry.
- A **Scaleway Managed Database for MySQL** for WooCommerce data persistence.
- Basic knowledge of Kubernetes, Terraform, WordPress, WooCommerce, and Scaleway's ecosystem.

Creating Terraform variables

1. Create a file `variables.tf` to define Terraform variables used in the project

Shell

```
variable "region" {
  type = string
  default = "fr-par"
  description = "The Scaleway region to deploy resources in (e.g., fr-par)"

}

variable "scaleway_access_key" {
  type = string
  description = "Scaleway access key for API authentication"
  sensitive = true

}
```

```
variable "scaleway_secret_key" {
  type = string
  description = "Scaleway secret key for API authentication"
  sensitive = true

}

variable "project_id" {
  type = string
  description = "Scaleway project ID for resource creation"

}

variable "db_user" {
  type = string
  description = "Username for the MySQL database"
  default = "admin"

}

variable "db_password" {
  type = string
  description = "Password for the MySQL database"
  sensitive = true
}
```

2. Use environment variables to securely provide your Scaleway credentials and sensitive data to Terraform. This avoids hardcoding values in files, reducing the risk of exposure (e.g., in version control).
   Set the required environment variables before running Terraform commands. Prefix the variable names with `TF_VAR_` to make them accessible to Terraform:

Shell

```
export TF_VAR_scaleway_access_key=""
export TF_VAR_scaleway_secret_key=""
export TF_VAR_project_id=""
export TF_VAR_db_password=""
# Optional: Override defaults if needed
export TF_VAR_region="fr-par"
```

```
export TF_VAR_db_user="admin"
```

Replace `<SCALEWAY_ACCESS_KEY>`, `<SCALEWAY_SECRET_KEY>`, `<SCALEWAY_PROJECT_ID>`, and `<MYSQL_DATABASE_PASSWORD>` with your actual values.

Best Practices:

- For persistent use on your local machine, add these export commands to your shell profile (e.g., `~/.bashrc` or `~/.zshrc`).
- In CI/CD environments, inject these as secure secrets.
- Never commit environment variable scripts or files containing credentials to version control.
- Verify variables are set (e.g., `echo $TF_VAR_scaleway_access_key` on Linux/macOS) before running Terraform.

Creating and configuring the Private Network and Public Gateway

Note: Ensure you have created a Virtual Private Cloud in the region of deployment. Learn more

Tip: Refer to the attached Terraform configuration file (main.tf) for full details.

1. Create a new Private Network named `shopsecure-private-net` in the `fr-par` region with a /22 IP subnet.

```
None
# Creating a private network for cluster isolation
resource "scaleway_vpc_private_network" "shopsecure_pn" {
  name = "shopsecure-private-net"
  region = var.region
}
```

2. Create a Public Gateway in the same region as the cluster:

```shell
# Creating a public gateway for external access
resource "scaleway_vpc_public_gateway" "shopsecure_gateway" {
  name = "shopsecure-gateway"
  type = "VPC-GW-S"
```

3. Attach the Public Gateway to the Private Network

```shell
# Attaching the Public Gateway to the Private Network with Dynamic NAT
resource "scaleway_vpc_gateway_network" "shopsecure_gateway_network" {
  gateway_id = scaleway_vpc_public_gateway.shopsecure_gateway.id
  private_network_id = scaleway_vpc_private_network.shopsecure_pn.id
  enable_masquerade = true
  ipam_config {
    push_default_route = true
  }
}
```

Creating a Kubernetes Kapsule cluster using Terraform

1. Create a Kubernetes Kosmos cluster with a node pool using full isolation. Configure the cluster via Terraform using the following characteristics:

- **Cluster Settings:**
  - Name: `shopsecure-kapsule-cluster`
  - Region: `fr-par`
  - Kubernetes Version: Latest supported (e.g., 1.32.7)
  - CNI: Cilium (recommended for performance and security)
  - Private Network: Attach to `shopsecure-private-net`
- **Pool Configuration**:
  - Pool Name: `full-isolation-pool`
  - Node Type: `PRO2-S` (4 vCPUs, 16GB RAM, suitable for WooCommerce workloads) Lean more about Instance types
  - Enable **Full Isolation**: Nodes will have only private IPs, relying on the Public Gateway for external communication.
  - Enable **Auto-scaling**: Set minimum nodes to 2 and maximum to 5 to handle traffic spikes.

- Enable **Autoheal**: Ensure unhealthy nodes are replaced automatically.

Terraform example:

```
None
# Creating a Kubernetes Kapsule cluster
resource "scaleway_k8s_cluster" "shopsecure_cluster" {
  name = "shopsecure-kapsule-cluster"
  version = "1.32.7"
  cni = "cilium"
  private_network_id = scaleway_vpc_private_network.shopsecure_pn.id
  region = var.region
  delete_additional_resources = true
}

# Creating a node pool with full isolation
resource "scaleway_k8s_pool" "full_isolation_pool" {
  cluster_id = scaleway_k8s_cluster.shopsecure_cluster.id
  name = "full-isolation-pool"
  node_type = "DEV1-M"
  size = 2
  autoscaling = true
  autohealing = true
  min_size = 2
  max_size = 5
  public_ip_disabled = true
  region = var.region
  depends_on = [scaleway_vpc_gateway_network.shopsecure_gateway_network]
}
```

Create and configure a Managed Database for MySQL using TerraForm

1. Create a Managed Database for MySQL that is being used for wooCommerce.

Terraform example:

```
None
# Creating a managed MySQL database for WooCommerce
resource "scaleway_rdb_instance" "woocommerce_db" {
```

```
  name = "shopsecure-woocommerce-db"
  node_type = "db-dev-s"
  engine = "MySQL-8"
  is_ha_cluster = true
  user_name = var.db_user
  password = var.db_password
  region = var.region
  private_network {
    pn_id = scaleway_vpc_private_network.shopsecure_pn.id
    enable_ipam = true
  }
}
```

2. Create a Kubernetes Secret for database credentials (woocommerce-db-secret.yaml):

```
None
apiVersion: v1
kind: Secret
metadata:
  name: woocommerce-db-credentials
  namespace: default
type: Opaque
data:
  DB_PASSWORD: "<base64-encoded-mysql-host>" # echo -n "mySecurePass123" | base64
# Ensure the DB_PASSWORD is the same password as set in the terraform.tfvars
configuration
```

Deploying the services using Terraform

1. Initialize your Terraform configuration: `terraform init`
2. Plan your deployment: `terraform plan`
3. Deploy the services: `terraform apply`

Create an Ingress controller

Deploy an Ingress controller (e.g., nginx-ingress) to manage external HTTP traffic to WooCommerce.

1. Create a configuration file `woocommerce-ingress.yaml`

```
None
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: shopsecure-ingress
  namespace: default
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
spec:
  rules:
  - host: shopsecure.your-domain.tld
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: shopsecure-woocommerce-service
            port:
              number: 80
```

Configure a Load Balancer

1. Create a configuration file `woocommerce-service.yaml` to deploy a Kubernetes Service of type `LoadBalancer`:

```
None
apiVersion: v1
kind: Service
metadata:
  name: shopsecure-woocommerce-service
  namespace: default
```

```
spec:
  selector:
    app: shopsecure-woocommerce
  ports:
  - port: 80
    targetPort: 80
  type: LoadBalancer
```

This configuration provisions a Scaleway Load Balancer will and integrates it with the Public Gateway to route external traffic.

Test and Monitor

- Download and Install the clusters `kubeconfig` file to interact with the cluster:

```
Shell
scw k8s config install <cluster-id>
```

- Deploy the web application using the following command:

```
Shell
kubectl apply -f woocommerce-db-secret.yaml
kubectl apply -f woocommerce-deployment.yaml
kubectl apply -f woocommerce-service.yaml
kubectl apply -f woocommerce-ingress.yaml
```

- Use `kubectl` to verify the cluster status:

```
Shell
kubectl get nodes
```

```
kubectl get pods
```

- Test the WooCommerce application by accessing `shopsecure.your-domain.tld` via a browser and verifying the WordPress setup with WooCommerce.
- Enable Scaleway Cockpit to monitor the cluster's data plane and audit logs.
- Verify auto-scaling by simulating increased traffic (e.g., using a load testing tool like Vegeta).

Tip: Once you have completed your tests, you can delete your deployment using the command `terraform destroy`

Benefits

- **Enhanced security**: Full isolation ensures that worker nodes are not exposed to the public internet, reducing attack surfaces. The Public Gateway provides controlled access with NAT.
- **Scalability**: Auto-scaling adjusts the number of nodes based on demand, ensuring cost efficiency and performance during traffic spikes (e.g., WooCommerce sales events).
- **High availability**: Deploying across multiple AZs (e.g., fr-par-1, fr-par-2) enhances resilience against zone failures.
- **Cost Efficiency**: Scaleway's free control plane and Private Network reduce costs, with only nodes and Load Balancers billed.
- **Simplified management**: Managed Kubernetes and Public Gateway abstract infrastructure complexities, allowing focus on WooCommerce store management.