**Project Documentation - Private VPC with SQL Database and GKE Cluster**

**Overview:**

This documentation provides a comprehensive overview of the project that involves setting up a private Virtual Private Cloud (VPC) with a SQL database and Google Kubernetes Engine (GKE) cluster.

Additionally, the project includes the deployment of a Django Todo sample application using Docker containers and Kubernetes.

**Project Structure:**

The project repository consists of the following files and folders:

**application:**

* This folder contains files related to the Django Todo sample application, including the application code, Dockerfile, and manifest files (deployment.yaml and service.yaml).

**modules:**

* This folder contains subfolders for different infrastructure components, such as Sqldatabase, vpc, vm, and gkecluster.
* Each subfolder contains the main.tf and variable.tf files for the respective infrastructure module.
* The VPC subfolder also includes the output.tf file.
* The module is called from the main configuration file using the module block, passing the required variables and getting the output from the module.

**sql.tf:**

* This file serves as the main configuration file for creating the SQL database and utilizes the modules defined in the modules/Sqldatabase folder.

**terraform.tfstate and terraform.tfstate.backup:**

* These files maintain the state of the Terraform-managed infrastructure.
* The .tfstate file contains the current state, and the .tfstate.backup file is a backup of the previous state.

**variable.tf:**

* This file defines the variables used throughout the Terraform configuration.

**vpc.tf:**

* This file serves as the main configuration file for creating the private VPC and utilizes the modules defined in the modules/vpc folder.

**gkecluster.tf:**

* This file serves as the main configuration file for creating the private GKE cluster and utilizes the modules defined in the modules/gkecluster folder.

**provider.tf:**

* This file defines the provider configurations for Terraform, specifying the necessary credentials and API endpoints for GCP.

**terraform:**

* The terraform binary, used to execute Terraform commands.

**Components Involved in Infrastructure Creation:**

**VPC (Virtual Private Cloud):**

* The code creates a private VPC network with a private subnet, establishes a service networking connection, sets up firewall rules for specific TCP ports, and configures NAT for private IP address translation.
* Each resource is defined with its specific attributes to achieve the desired network setup in GCP.
* The google\_compute\_network.vpc resource creates a VPC network named "vpc".
* The google\_compute\_subnetwork.private\_subnet resource creates a private subnet named "sub-pri" within the VPC network.
* The google\_compute\_global\_address.private\_ip\_block resource creates a global address named "private-ip-block" for VPC peering.
* The google\_service\_networking\_connection.private\_vpc\_connection resource establishes a service networking connection between the VPC network and "servicenetworking.googleapis.com" service.
* The google\_compute\_firewall.default resource creates a firewall rule named "default" that allows TCP traffic on ports 80, 5432, and 3306 from the 10.0.0.0/16 CIDR block to the VPC network.
* The google\_compute\_address.ip\_address resource creates an internal IP address named "private-ip" with the address 10.0.42.42 in the "sub-pri" subnet.
* The google\_compute\_router.nat\_router resource creates a NAT router named "nat-router" in the VPC network.
* The google\_compute\_router\_nat.nat\_config resource creates a NAT configuration named "nat-config" for the "nat-router" router. The NAT configuration allows all IP ranges in all subnets in the VPC network to be translated to the NAT router's public IP address.

**SQL Database**

* The code creates various resources in Google Cloud Platform (GCP) to set up a private SQL database instance with a database and a user.
* The google\_sql\_database.database resource creates a database named "my-database" in the "tricustom" SQL database instance.
* The google\_sql\_database\_instance.tricustom resource creates a SQL database instance named "tricustom" in the "us-central1" region. The deletion\_protection attribute is set to false, ensuring that the database instance cannot be deleted accidentally.
* The settings block of the google\_sql\_database\_instance.tricustom resource specifies the configuration for the database instance. The tier attribute specifies the tier of the database instance, such as "db-f1-micro" or "db-n1-standard-1". The disk\_size attribute specifies the size of the database instance's disk in GB. The availability\_type attribute specifies the availability type of the database instance, which can be "REGIONAL" or "ZONAL".
* The ip\_configuration block of the google\_sql\_database\_instance.tricustom resource specifies the IP configuration for the database instance. The ipv4\_enabled attribute is set to false, indicating that the database instance will not have a public IP address. The private\_network attribute specifies the VPC network that the database instance will be in.
* The google\_sql\_user.db\_users resource creates a SQL user named "tricustom" in the "tricustom" SQL database instance. The password attribute specifies the password for the SQL user.

**GKE Cluster:**

The code creates a Google Kubernetes Engine (GKE) cluster with the following configurations:

**Name:** tricustom\_cluster

**Initial Node Count:** Value provided by the variable var.node\_count

**Master Authentication**: Client certificates will not be issued

**Network:** The cluster will be created in the VPC specified by var.vpc

**Subnetwork:** The cluster will be created in the subnetwork specified by var.privatesub

**Private Cluster:** The cluster will be private, with a private endpoint for the Kubernetes master and private nodes.

**Master Authorized Networks:** Only the CIDR block 10.0.0.0/16 will be allowed to access the Kubernetes master.

**IP Allocation Policy:** The CIDR block 10.4.0.0/14 will be used for cluster IP addresses, and 192.168.0.0/20 will be used for service IP addresses.

**The output resources are as follows:**

output.kubeconfig\_path: This output provides the path to the Kubernetes configuration file for the cluster. The configuration file allows users to interact with the cluster from a client.

output.cluster\_endpoint: This output provides the endpoint for the Kubernetes master. The endpoint can be used to access the Kubernetes API from a client.

**VM (Google Compute Engine Instance):**

* The code provided creates a Google Compute Engine (GCE) instance named "webserver" in the "us-central1-a" zone.
* The machine\_type attribute of the google\_compute\_instance.tricustomvm1 resource specifies the machine type for the instance.
* The boot\_disk block specifies the boot disk for the instance.
* The initialize\_params block of the boot\_disk block specifies the parameters for the boot disk.
* The image attribute specifies the image that the boot disk will be initialized with.
* The network\_interface block specifies the network interface for the instance.
* The subnetwork attribute specifies the subnetwork that the network interface will be in.

**Load Balancer:**

* The code uses a LoadBalancer service to expose the Django Todo application from the private subnet.
* The application is deployed using Kubernetes with a LoadBalancer service configured to listen on port 80.
* The LoadBalancer allows external access to the application running in the private GKE cluster.

**Project Workflow:**

The project workflow can be summarized as follows:

**VPC Creation:**

* The project begins with the creation of a private Virtual Private Cloud (VPC) using Terraform.
* The vpc.tf file leverages the modules defined in the modules/vpc folder to set up the VPC infrastructure.
* The private VPC ensures isolation and security of the resources deployed within it.

**SQL Database:**

* The SQL database instance is created within the private VPC.
* The sql.tf file utilizes the modules defined in the modules/Sqldatabase folder to manage the SQL database resources.
* The SQL database stores the necessary data for the Django Todo application.

**GKE Cluster:**

* A private Google Kubernetes Engine (GKE) cluster is set up within the private VPC.
* The gkecluster.tf file uses the modules defined in the modules/gkecluster folder to create and manage the GKE resources.
* The private GKE cluster ensures a secure and isolated environment for running containerized applications.

**Django Todo Application Deployment:**

* The Django Todo sample application is containerized using Docker and stored in a container registry.
* The application code and Dockerfile are located in the application folder.
* The deployment.yaml and service.yaml manifest files define the Kubernetes deployment and service configurations.
* A LoadBalancer service is exposed on port 80 to allow external access to the application.

**Why Django chosen for this project:**

**Full-Featured Web Framework:** Django is a powerful and full-featured web framework that comes with built-in components for handling common web development tasks, such as URL routing, form handling, authentication, and database management. This feature-rich nature of Django allows developers to build applications quickly and efficiently.

**Batteries-Included Approach:** Django follows a "batteries-included" philosophy, meaning it provides a wide range of built-in tools and libraries, reducing the need for external dependencies.

This can be beneficial for rapid development as developers don't need to spend time searching for and integrating third-party libraries for common functionalities.

**Security**: Django has a strong focus on security and includes built-in features to protect against common web application vulnerabilities like SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF). By default, Django's ORM (Object-Relational Mapping) protects against SQL injection, making it safer to work with databases.

**Scalability:** Django is designed to handle scalable applications. Its architecture allows for modular design and the use of caching mechanisms, which helps in optimizing performance and handling large user bases.

**Documentation and Community Support:** Django has extensive and well-maintained documentation, making it easier for developers to learn and troubleshoot. Additionally, Django has a large and active community that provides support, resources, and third-party packages.

**Versatility**: Django is not limited to web applications; it can be used to build APIs, web services, and even perform background tasks using Celery.

**ORM and Database Support:** Django's built-in Object-Relational Mapping (ORM) allows developers to work with databases using Python objects, making database management more straightforward and less error-prone. Django supports multiple databases, including PostgreSQL, MySQL, SQLite, and Oracle.

**Admin Interface:** Django comes with a built-in admin interface that enables developers to manage application data without building a custom admin panel separately.

**Maturity and Stability:** Django has been around for many years and has been used to build numerous high-profile websites and applications. Its maturity and stability make it a reliable choice for production-ready projects.

**Security Measures:**

**Private VPC and Subnetworks:**

* The project sets up a private Virtual Private Cloud (VPC) network.
* Private networks isolate resources from the public internet, reducing exposure to external threats.
* Subnetworks are created within the VPC to segment and isolate resources further.
* This helps control access and limits the impact of potential security breaches.

**Firewall Rules and Security Groups:**

* Firewall rules are configured to control incoming and outgoing traffic. In the project, a firewall rule named "default" is created to allow specific TCP traffic on ports 80, 5432, and 3306 from a specific IP range (e.g., 10.0.0.0/16) to the VPC network.
* Google Cloud Platform (GCP) allows the use of security groups, which provide additional security controls at the instance level to control traffic.

**Private IP Addressing and NAT:**

* Private IP addressing is used for instances within the VPC, which means they are not directly accessible from the public internet.
* Network Address Translation (NAT) is employed to allow instances within the VPC to access the internet while hiding their internal private IP addresses.

**Service Networking Connection:**

* A service networking connection is established between the VPC network and the "servicenetworking.googleapis.com" service.
* This allows the secure communication between the VPC network and specific Google services.

**SQL Database Security:**

* The SQL database instance is created within the private VPC, making it accessible only from within the VPC network, enhancing security.
* SQL users are created with proper access controls and secure passwords to ensure data access is limited to authorized users.

**GKE Cluster Security:**

* The GKE cluster is set up within the private VPC, making the Kubernetes master and nodes accessible only from within the VPC network.
* Master Authorized Networks are configured to allow only the specified CIDR block (e.g., 10.0.0.0/16) to access the Kubernetes master.

**Continuous Security Review:**

* Regular security audits and reviews of the infrastructure and applications are conducted to identify and address any security vulnerabilities or misconfigurations.

**CI/CD pipeline using Jenkins:**

**Install Jenkins:**

* Download and install Jenkins on your server or cloud instance following the official installation guide for your operating system or cloud platform.

**Access Jenkins Web Interface:**

* Once Jenkins is installed, access the Jenkins web interface using your browser. The default address is usually http://localhost:8080/.

**Install Required Plugins:**

* In Jenkins, navigate to "Manage Jenkins" > "Manage Plugins."
* Install the necessary plugins for version control systems (e.g., Git), build tools (e.g., Maven), and other requirements. The suggested plugins can be installed during initial setup.

**Create a New Jenkins Job:**

* From the Jenkins dashboard, click on "New Item" to create a new job.
* Choose "Freestyle project" to set up a simple CI/CD pipeline.

**Configure Source Code Management:**

* Under the job configuration, find the "Source Code Management" section.
* Choose your version control system (e.g., Git) and provide the repository URL and credentials if required.

**Define Build Steps:**

* In the job configuration, find the "Build" section and add build steps.
* For example, you can use a "Execute shell" build step to run commands, such as fetching dependencies, running tests, and building the project.

**Set Up Post-Build Actions:**

* Under the job configuration, find the "Post-build Actions" section.
* Add actions like "Archive the artifacts" to keep build artifacts for future use or "Email Notification" to receive build status notifications.

**Save the Job and Run It Manually:**

* Save the job configuration, and you'll be taken back to the Jenkins dashboard.
* Run the job manually by clicking on the "Build Now" button to test if the CI pipeline works as expected.

**Set Up Deployment (Optional):**

* If you need to set up continuous deployment, you can extend the job configuration or create additional jobs for deployment purposes.
* Use plugins or custom scripts to deploy the build artifacts to the desired environment.

**Configure Automatic Triggers (Optional):**

* For automatic triggering of the CI/CD pipeline on code changes, set up webhooks or configure Jenkins to poll your version control system for changes.

**Conclusion:**

* The project successfully demonstrates the creation of a private VPC with a SQL database and GKE cluster. Additionally, it deploys a Django Todo sample application using Docker containers and Kubernetes.
* The use of Terraform enables infrastructure-as-code practices, making it easy to manage and scale the cloud infrastructure while ensuring proper security controls.
* The project's components work together to provide a robust and secure environment for hosting applications in a private network.