TASK_6
PROBLEM_STATEMENT

Deploy the Wordpress application on Kubernetes and AWS using terraform including the following steps;

1. Write an Infrastructure as code using terraform, which automatically deploy the Wordpress application
2. On AWS, use RDS service for the relational database for Wordpress application.
3. Deploy the Wordpress as a container either on top of Minikube or EKS or Fargate service on AWS
4. The Wordpress application should be accessible from the public world if deployed on AWS or through workstation if deployed on Minikube.

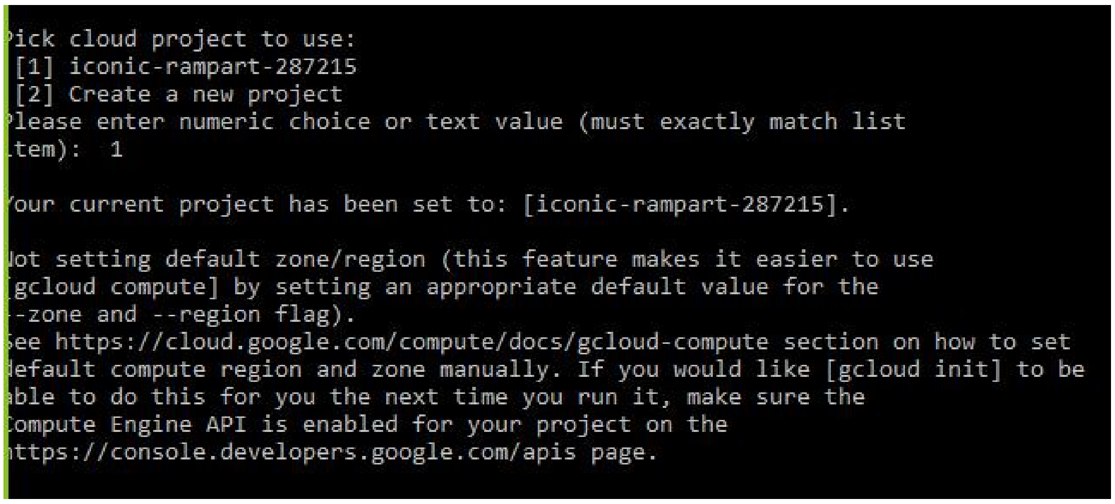
Try each step first manually and write Terraform code for the same to have a proper understanding of workflow of task.

Step - 1: First of all, configure your AWS profile.

```
aws configure --profile Chaitanya

AWS Access Key ID [****************NOP3]:

AWS Secret Access Key [****************CKLD]:

Default region name [ap-south-1]:

Default output format [None]:
```

Step - 2: download the Google Cloud SDK & login through your credentials in the GCP Cloud.

```
provider "google" {

 credentials = file("${var.gcp_credentials_path}")

 project     = var.gcp_project_id

 region      = var.gcp_cur_region

}
```



```
ick cloud project to use:
 [1] iconic-rampart-287215
 [2] Create a new project
lease enter numeric choice or text value (must exactly match list
tem):  1

our current project has been set to: [iconic-rampart-287215].

ot setting default zone/region (this feature makes it easier to use
gcloud compute] by setting an appropriate default value for the
-zone and --region flag).
ee https://cloud.google.com/compute/docs/gcloud-compute section on how to set
efault compute region and zone manually. If you would like [gcloud init] to be
ble to do this for you the next time you run it, make sure the
ompute Engine API is enabled for your project on the
ttps://console.developers.google.com/apis page.
```

Step - 3: Create a **var.tf** file in which we will store all the variables that we need to use in our code.

```
variable "gcp_credentials_path"{


    default="C:\\Users\\Chait\\AppData\\Roaming\\gcloud\\MyProject-a04023bd5af9.json"

 }



 variable "gcp_project_id"{



     default="iconic-rampart-3567281"

 }



 variable "gcp_cur_region"{



      default="asia-south1"

 }



 variable "aws_profile"{


    default="Chaitanya"

 }



 variable "aws_region"{


   default= "ap-south-1"

 }


 variable "gcp_vpc_name"{


    default = "gcp-vpc"


 }


 variable "subnet_gcp_name"{

    default = "subnet-vpc"

 }


 variable "subnet_ip_cidr_range"{

    default = "10.0.2.0/24"
```

```
}

variable "gcp_subnet_region"{
    default = "asia-southeast1"
}

variable "gcp_compute_firewall"{
    default = "firewall-gcp"
}

variable "allowed_ports"{


    type=list
    default=["80","22"]


}



variable "google_container_cluster_name"{


    default="gcp-cluster"
}

variable "google_container_cluster_location"{
    default = "asia-southeast1"
}

variable "gcp_node_config_machine_type"{
    default = "n1-standard-1"
}



variable "aws_db_instance_storage_type"{
    default = "gp2"
}

variable "aws_db_instance_engine"{
default = "mysql"
}

variable "aws_db_instance_engine_version"{
default = 5.7
}

variable "aws_db_instance_instance_class"{
```

```
default = "db.t2.micro"

}


variable "aws_db_instance_db_name"{

default = "db"

}


variable "aws_db_instance_username"{

default = "admin"

}


variable "aws_db_instance_password"{

default = "chaitanya"

}


variable "aws_db_instance_publicly_accessible"{

default = true

}


variable "aws_db_instance_skip_final_snapshot"{

default = true

}
```

Step - 3: Now we will create main.tf file

```
module "gcp_aws"{



  source = "./modules"



  gcp_project_id=var.gcp_project_id

  gcp_vpc_name=var.gcp_vpc_name

  subnet_gcp_name=var.subnet_gcp_name

  subnet_ip_cidr_range=var.subnet_ip_cidr_range

  gcp_subnet_region=var.gcp_subnet_region

  gcp_compute_firewall=var.gcp_compute_firewall

  allowed_ports=var.allowed_ports





  google_container_cluster_name=var.google_container_cluster_name

  google_container_cluster_location=var.google_container_cluster_location

  gcp_node_config_machine_type=var.gcp_node_config_machine_type





  aws_db_instance_storage_type=var.aws_db_instance_storage_type
```

```
    aws_db_instance_engine=var.aws_db_instance_engine

    aws_db_instance_engine_version=var.aws_db_instance_engine_version

    aws_db_instance_instance_class=var.aws_db_instance_instance_class

    aws_db_instance_db_name=var.aws_db_instance_db_name

    aws_db_instance_username=var.aws_db_instance_username

    aws_db_instance_password=var.aws_db_instance_password

    aws_db_instance_publicly_accessible=var.aws_db_instance_publicly_accessible

    aws_db_instance_skip_final_snapshot=var.aws_db_instance_skip_final_snapshot


    }
```

Step - 4: Now, we need to create a VPC, Subnet & Firewall in GCP.

```
variable "gcp_vpc_name"{}

    variable "subnet_gcp_name"{}

    variable "subnet_ip_cidr_range"{}

    variable "gcp_subnet_region"{}

    variable "gcp_compute_firewall"{}

    variable "allowed_ports"{}

    variable "gcp_project_id"{}



    // Creating a VPC

    resource "google_compute_network" "vpc_gcp" {

     name =  var.gcp_vpc_name

     auto_create_subnetworks=false

      project= var.gcp_project_id

    }

    // Creating a subnetwork

    resource "google_compute_subnetwork" "subnet_vpc" {

        depends_on=[google_compute_network.vpc_gcp]

        name         =var.subnet_gcp_name

     ip_cidr_range = var.subnet_ip_cidr_range

     region       =var.gcp_subnet_region

     network      = google_compute_network.vpc_gcp.id

    }

    // Creating a firewall

    resource "google_compute_firewall" "default" {

    depends_on=[google_compute_network.vpc_gcp]

     name    =var.gcp_compute_firewall

     network = google_compute_network.vpc_gcp.name

     allow {

           protocol = "icmp"

     }

     allow {

           protocol = "tcp"
```

```
          ports    = var.allowed_ports

    }

   }
```

Step - 5: Now, we launch GKE cluster.

```
variable "google_container_cluster_name"{}


     variable "google_container_cluster_location"{}


     variable "gcp_node_config_machine_type"{}



     resource "google_container_cluster" "gcp_cluster" {

     depends_on=[google_compute_network.vpc_gcp]

      name              = var.google_container_cluster_name

      location          = var.google_container_cluster_location

      initial_node_count = 1

      master_auth {

            username = ""

            password = ""

            client_certificate_config {

               issue_client_certificate = false

            }

        }

        node_config {

            machine_type= "n1-standard-1"

        }

        network= google_compute_network.vpc_gcp.name

        project=var.gcp_project_id

        subnetwork=google_compute_subnetwork.subnet_vpc.name

     }

     // running the command to update the kubeconfig file

     resource "null_resource" "cluster" {

     provisioner "local-exec" {

      command ="gcloud container clusters get-credentials ${google_container_cluster.gcp_cluster.name}  --region
${google_container_cluster.gcp_cluster.location} --project ${google_container_cluster.gcp_cluster.project}"

      }

     }
```

Step - 6: Now, we need to launch our RDS database in AWS cloud.

```
 variable "aws_db_instance_storage_type"{}


   variable "aws_db_instance_engine"{}
```

```
variable "aws_db_instance_engine_version"{}


variable "aws_db_instance_instance_class"{}


variable "aws_db_instance_db_name"{}


variable "aws_db_instance_username"{}


variable "aws_db_instance_password"{}


variable "aws_db_instance_publicly_accessible"{}


variable "aws_db_instance_skip_final_snapshot"{}



resource "aws_vpc" "defaultvpc" {
           cidr_block = "192.168.0.0/16"
           instance_tenancy = "default"
           enable_dns_hostnames = true
           tags = {
             Name = "lw_vpc"
           }
         }



resource "aws_subnet" "lw_public_subnet" {
           vpc_id = aws_vpc.defaultvpc.id
           cidr_block = "192.168.0.0/24"
           availability_zone = "ap-south-1a"
           map_public_ip_on_launch = "true"
           tags = {
             Name = "lw_public_subnet"
           }
         }


resource "aws_subnet" "lw_public_subnet2" {
           vpc_id = aws_vpc.defaultvpc.id
           cidr_block = "192.168.1.0/24"
           availability_zone = "ap-south-1b"
           map_public_ip_on_launch = "true"
           tags = {
             Name = "lw_public_subnet2"
           }
         }
```

```
resource "aws_db_subnet_group" "default" {
  name      = "main"
  subnet_ids = [aws_lw.sparsh_public_subnet.id,aws_subnet.lw_public_subnet2.id]

  tags = {
    Name = "subnet"
  }
}

resource "aws_internet_gateway" "lw_gw" {
            vpc_id = aws_vpc.defaultvpc.id
            tags = {
              Name = "lw_gw"
            }
          }



resource "aws_security_group" "lw_public_sg" {
            depends_on=[google_container_cluster.gcp_cluster]
            name       = "HTTP_SSH_PING"
            description = "It allows HTTP SSH PING inbound traffic"
            vpc_id     = aws_vpc.defaultvpc.id

 ingress {
    description = "TLS from VPC"
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
    ipv6_cidr_blocks = ["::/0"]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
    ipv6_cidr_blocks = ["::/0"]
  }



              tags = {
              Name = "HTTP_SSH_PING"
            }
          }



   resource "aws_db_instance" "wp_db" {
      depends_on=[aws_security_group.lw_public_sg]
         allocated_storage    = 20
         storage_type         = var.aws_db_instance_storage_type
         engine               = var.aws_db_instance_engine
         engine_version       = var.aws_db_instance_engine_version
         instance_class       = var.aws_db_instance_instance_class
         name                 = var.aws_db_instance_db_name
         username             = var.aws_db_instance_username
         password             = var.aws_db_instance_password
         parameter_group_name = "default.mysql5.7"
         publicly_accessible  = var.aws_db_instance_publicly_accessible
         skip_final_snapshot  = var.aws_db_instance_skip_final_snapshot
         vpc_security_group_ids = [aws_security_group.sparsh_public_sg.id]
         db_subnet_group_name = aws_db_subnet_group.default.name
    }
```

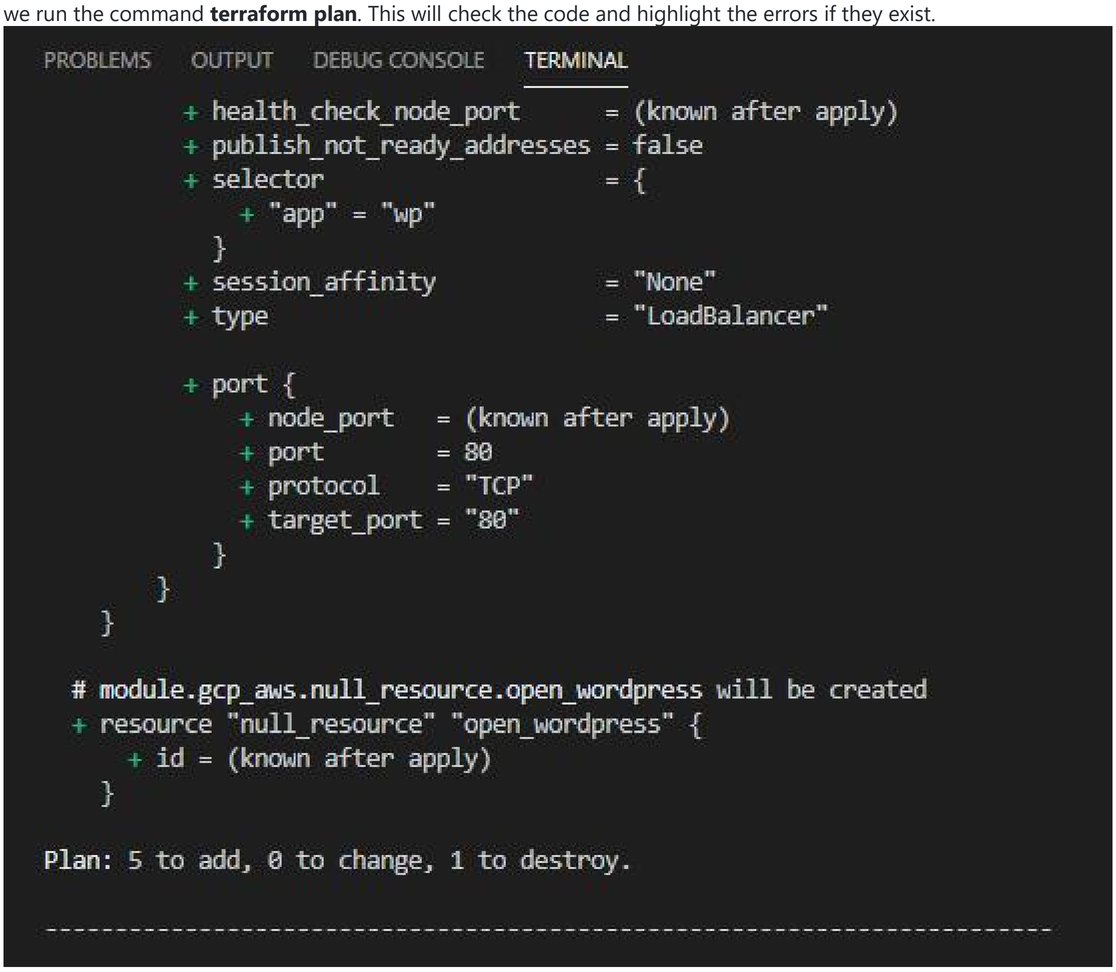
Step - 7: Now launch WordPress server on top of GKE cluster using terraform.

```
 provider "kubernetes" {

config_context_cluster="gke_${google_container_cluster.gcp_cluster.project}_${google_container_cluster.gcp_cluster.location}_${google_container_cluster.gcp_cluster.name}"
      }
      resource "kubernetes_service" "k8s" {
         depends_on=[aws_db_instance.wp_db,google_container_cluster.gcp_cluster]
         metadata{
            name="wp"
            labels={
               env="test"
               name="wp"
            }
         }
         spec{
            type="LoadBalancer"
            selector={
            app="wp"
            }
            port{
               port=80
               target_port=80
            }
         }
```

```
}

output "ip_add"{

    value=kubernetes_service.k8s.load_balancer_ingress[0].ip

}

resource "kubernetes_deployment" "wp_deploy"{

    depends_on=[aws_db_instance.wp_db,google_container_cluster.gcp_cluster]

    metadata{

        name="wp-deploy"

        labels={

            name="wp-deploy"

            app="wp"

        }

    }

    spec{

        replicas=1

        selector{

            match_labels = {

                app="wp"

            }

        }

        template{

            metadata{

                name="wp-deploy"

                labels={

                    app="wp"

                }

            }

            spec{

                container{

                    name="wp"

                    image="wordpress"

                    env{

                        name="WORDPRESS_DB_HOST"

                        value=aws_db_instance.wp_db.address

                    }

                    env{

                        name="WORDPRESS_DB_USER"

                        value=aws_db_instance.wp_db.username

                    }

                    env{

                        name="WORDPRESS_DB_PASSWORD"

                        value=aws_db_instance.wp_db.password

                    }

                    env{

                        name="WORDPRESS_DB_NAME"
                        value=aws_db_instance.wp_db.name

                    }

                }

            }

        }

    }

}

// open wordpress site in browser

resource "null_resource" "open_wordpress" {

provisioner "local-exec" {

command ="start chrome ${kubernetes_service.k8s.load_balancer_ingress[0].ip}"

    }

}
```

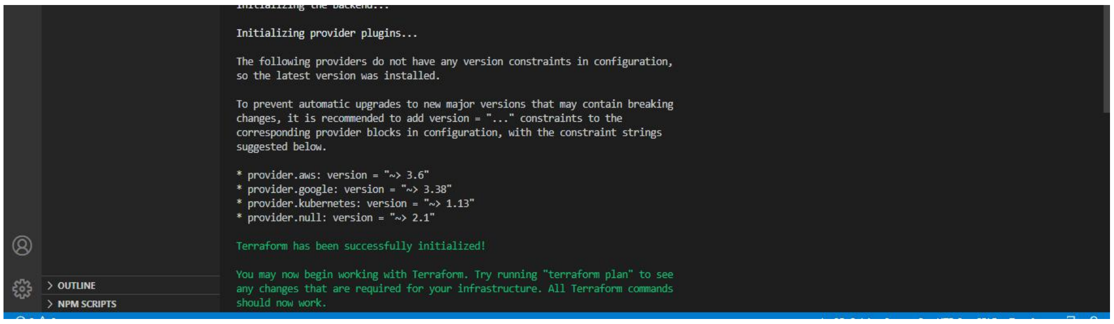
Step - 8: Now, we run our terraform code.

we run the command **terraform plan**. This will check the code and highlight the errors if they exist.



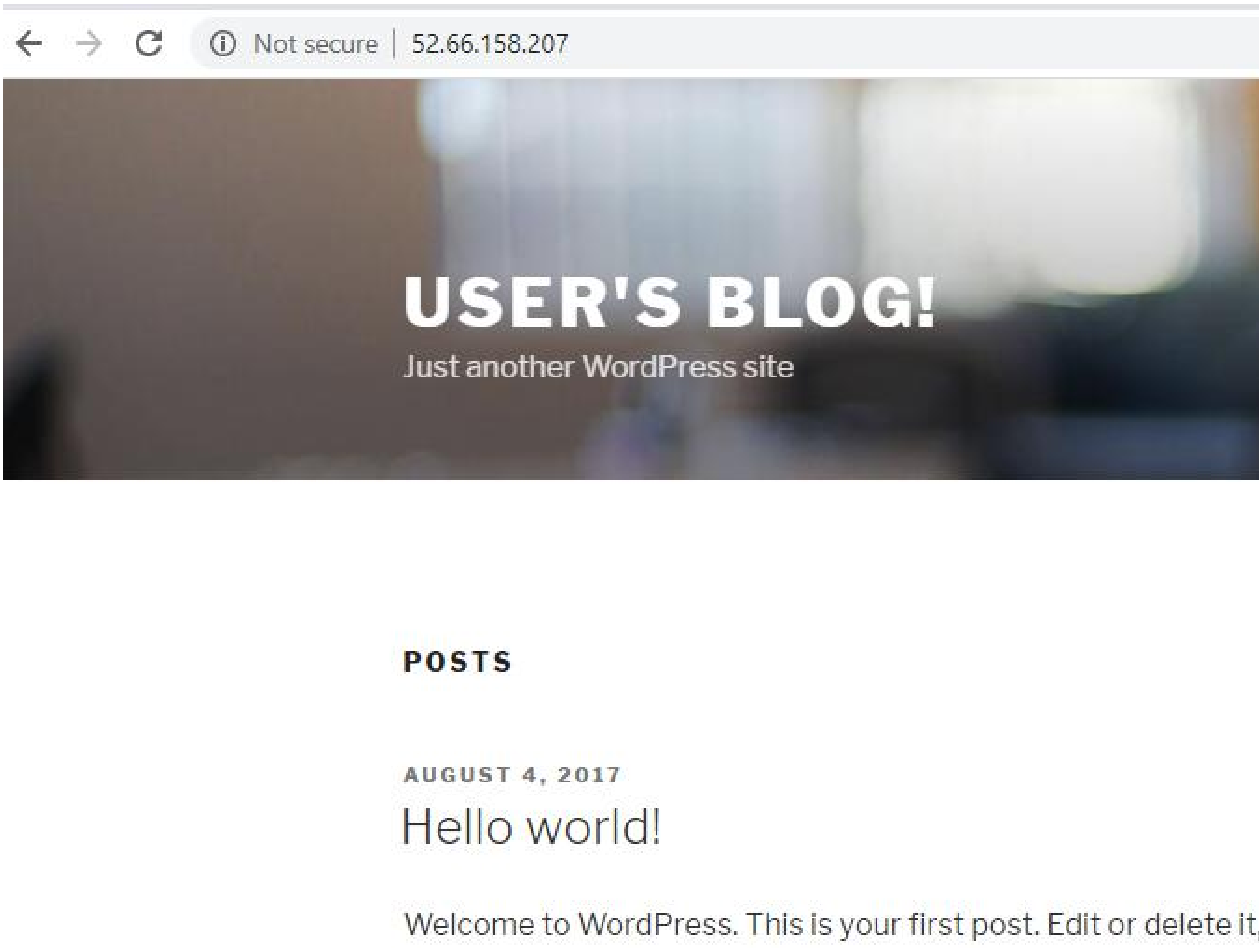
```
          + health_check_node_port      = (known after apply)
          + publish_not_ready_addresses = false
          + selector                    = {
              + "app" = "wp"
            }
          + session_affinity            = "None"
          + type                        = "LoadBalancer"

          + port {
              + node_port   = (known after apply)
              + port        = 80
              + protocol    = "TCP"
              + target_port = "80"
            }
        }
    }

  # module.gcp_aws.null_resource.open_wordpress will be created
  + resource "null_resource" "open_wordpress" {
      + id = (known after apply)
    }

Plan: 5 to add, 0 to change, 1 to destroy.

------------------------------------------------------------------------
```

Finally, we run the command **terraform apply --auto-approve**.



```
Initializing the backend...

Initializing provider plugins...

The following providers do not have any version constraints in configuration,
so the latest version was installed.

To prevent automatic upgrades to new major versions that may contain breaking
changes, it is recommended to add version = "..." constraints to the
corresponding provider blocks in configuration, with the constraint strings
suggested below.

* provider.aws: version = "~> 3.6"
* provider.google: version = "~> 3.38"
* provider.kubernetes: version = "~> 1.13"
* provider.null: version = "~> 2.1"

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.
```

NOW OUR WEBSITE IS DEPLOYED



USER'S BLOG!

Just another WordPress site

POSTS

AUGUST 4, 2017

Hello world!

Welcome to WordPress. This is your first post. Edit or delete it