This document has the coding for the 3 Tier application setup in Google Cloud

./main.tf

Enabling Services:

We need to enable the following services:

* **Service Networking & Serverless VPC Access** -
* **Cloud Build**
* **Cloud Memorystore**
* **Cloud Run** .
* **Cloud SQL**
* **Cloud Storage**
* **Cloud Secret Manager**
* **Artifact Registry**

variable "gcp\_service\_list" {  
    description = "The list of apis necessary for the project"  
    type        = list(string)  
    default = [  
        "compute.googleapis.com",  
        "cloudapis.googleapis.com",  
        "vpcaccess.googleapis.com",  
        "servicenetworking.googleapis.com",  
        "cloudbuild.googleapis.com",  
        "sql-component.googleapis.com",  
        "sqladmin.googleapis.com",  
        "storage.googleapis.com",  
        "secretmanager.googleapis.com",  
        "run.googleapis.com",  
        "artifactregistry.googleapis.com",  
        "redis.googleapis.com"  
    ]  
}  
  
resource "google\_project\_service" "all" {  
    for\_each                   = toset(var.gcp\_service\_list)  
    project                    = var.project\_number  
    service                    = each.key  
    disable\_on\_destroy = false  
}

**Set permissions**

The following command sets IAM Roles and Permissions that allow Cloud Build to deploy services.

Enable Cloud Build Service Account to deploy to Cloud Run

Enable Cloud Build Service Account to set VPN Access for Cloud Run

Enable Cloud Build Service Account to perform Service Account activities

Enable Cloud Build Service Account to act on behalf of Compute Service Account

Enable Cloud Build Service Account to publish to Cloud Run

Enable Cloud Build Service Account to consume secrets

Enable Cloud Build Service Account to store containers in Artifact Registry

variable "build\_roles\_list" {  
    description = "The list of roles that build needs for"  
    type        = list(string)  
    default = [  
        "roles/run.developer",  
        "roles/vpaccess.user",  
        "roles/iam.serviceAccountUser",  
        "roles/run.admin",  
        "roles/secretmanager.secretAccessor",  
        "roles/artifactregistry.admin",  
    ]  
}  
resource "google\_project\_iam\_member" "allbuild" {  
    for\_each   = toset(var.build\_roles\_list)  
    project    = var.project\_number  
    role       = each.key  
    member     = "serviceAccount:${local.sabuild}"  
    depends\_on = [google\_project\_service.all]  
}

**Create networking for SQL instance**

resource "google\_compute\_global\_address" "google\_managed\_services\_vpn\_connector" {  
    name          = "google-managed-services-vpn-connector"  
    purpose       = "VPC\_PEERING"  
    address\_type  = "INTERNAL"  
    prefix\_length = 16  
    network       = local.defaultnetwork  
    project       = var.project\_id  
    depends\_on    = [google\_project\_service.all]  
}  
resource "google\_service\_networking\_connection" "vpcpeerings" {  
    network                 = local.defaultnetwork  
    service                 = "servicenetworking.googleapis.com"  
    reserved\_peering\_ranges = [google\_compute\_global\_address.google\_managed\_services\_vpn\_connector.name]  
}

**Create VPC access connector**

resource "google\_vpc\_access\_connector" "connector" {  
    provider      = google-beta  
    project       = var.project\_id  
    name          = "vpc-connector"  
    ip\_cidr\_range = "10.8.0.0/28"  
    network       = "default"  
    region        = var.region  
    depends\_on    = [google\_compute\_global\_address.google\_managed\_services\_vpn\_connector, google\_project\_service.all]  
}

**Create Redis Server**

resource "google\_redis\_instance" "todo\_cache" {  
    authorized\_network      = local.defaultnetwork  
    connect\_mode            = "DIRECT\_PEERING"  
    location\_id             = var.zone  
    memory\_size\_gb          = 1  
    name                    = "${var.basename}-cache"  
    project                 = var.project\_id  
    redis\_version           = "REDIS\_6\_X"  
    region                  = var.region  
    reserved\_ip\_range       = "10.137.125.88/29"  
    tier                    = "BASIC"  
    transit\_encryption\_mode = "DISABLED"  
    depends\_on              = [google\_project\_service.all]  
}

**Create SQL Server**

resource "google\_sql\_database\_instance" "todo\_database" {  
    name="${var.basename}-db-${random\_id.id.hex}"  
    database\_version = "MYSQL\_5\_7"  
    region           = var.region  
    project          = var.project\_id  
    settings {  
        tier                  = "db-g1-small"  
        disk\_autoresize       = true  
        disk\_autoresize\_limit = 0  
        disk\_size             = 10  
        disk\_type             = "PD\_SSD"  
        ip\_configuration {  
            ipv4\_enabled    = false  
            private\_network = local.defaultnetwork  
        }  
        location\_preference {  
            zone = var.zone  
        }  
    }  
    deletion\_protection = false  
    depends\_on = [  
        google\_project\_service.all,  
        google\_service\_networking\_connection.vpcpeerings  
    ]  
    # This handles loading the schema after the database installs.  
    provisioner "local-exec" {  
        working\_dir = "${path.module}/code/database"  
        command     = "./load\_schema.sh ${var.project\_id} ${google\_sql\_database\_instance.todo\_database.name}"  
    }  
}

**Create Artifactory Registry Repository**

resource "google\_artifact\_registry\_repository" "todo\_app" {  
    provider      = google-beta  
    format        = "DOCKER"  
    location      = var.region  
    project       = var.project\_id  
    repository\_id = "${var.basename}-app"  
    depends\_on    = [google\_project\_service.all]  
}

**Create Secrets**

resource "google\_secret\_manager\_secret" "redishost" {  
    project = var.project\_number  
    replication {  
        automatic = true  
    }  
    secret\_id  = "redishost"  
    depends\_on = [google\_project\_service.all]  
}  
resource "google\_secret\_manager\_secret\_version" "redishost" {  
    enabled     = true  
    secret      = "projects/${var.project\_number}/secrets/redishost"  
    secret\_data = google\_redis\_instance.todo\_cache.host  
    depends\_on  = [google\_project\_service.all, google\_redis\_instance.todo\_cache, google\_secret\_manager\_secret.redishost]  
}  
resource "google\_secret\_manager\_secret" "sqlhost" {  
    project = var.project\_number  
    replication {  
        automatic = true  
    }  
    secret\_id  = "sqlhost"  
    depends\_on = [google\_project\_service.all]  
}  
resource "google\_secret\_manager\_secret\_version" "sqlhost" {  
    enabled     = true  
    secret      = "projects/${var.project\_number}/secrets/sqlhost"  
    secret\_data = google\_sql\_database\_instance.todo\_database.private\_ip\_address  
    depends\_on  = [google\_project\_service.all, google\_sql\_database\_instance.todo\_database, google\_secret\_manager\_secret.sqlhost]  
}

**Create Artifact for middleware**

The following command creates the Docker image and hosts it on Artifact Registry: [./code/frontend/clouldbuild.yaml](https://cloud.google.com/shell/docs/cloud-shell-tutorials/deploystack/three-tier-app#fe-cloudbuild)

resource "null\_resource" "cloudbuild\_api" {  
  provisioner "local-exec" {  
    working\_dir = "${path.module}/code/middleware"  
    command     = "gcloud builds submit . --substitutions=\_REGION=${var.region},\_BASENAME=${var.basename}"  
  }  
  depends\_on = [  
    google\_artifact\_registry\_repository.todo\_app,  
    google\_secret\_manager\_secret\_version.redishost,  
    google\_secret\_manager\_secret\_version.sqlhost,  
    google\_project\_service.all  
  ]  
}

**Deploy API container to Cloud Run**

resource "google\_cloud\_run\_service" "api" {  
    name     = "${var.basename}-api"  
    location = var.region  
    project  = var.project\_id  
  
    template {  
        spec {  
            containers {  
                image = "${var.region}-docker.pkg.dev/${var.project\_id}/${var.basename}-app/api"  
                env {  
                    name = "REDISHOST"  
                    value\_from {  
                        secret\_key\_ref {  
                            name = google\_secret\_manager\_secret.redishost.secret\_id  
                            key  = "latest"  
                        }  
                    }  
                }  
                env {  
                    name = "todo\_host"  
                    value\_from {  
                        secret\_key\_ref {  
                        name = google\_secret\_manager\_secret.sqlhost.secret\_id  
                        key  = "latest"  
                        }  
                    }  
                }  
                env {  
                    name  = "todo\_user"  
                    value = "todo\_user"  
                }  
                env {  
                    name  = "todo\_pass"  
                    value = "todo\_pass"  
                }  
                env {  
                    name  = "todo\_name"  
                    value = "todo"  
                }  
                env {  
                    name  = "REDISPORT"  
                    value = "6379"  
                }  
            }  
        }  
        metadata {  
            annotations = {  
                "autoscaling.knative.dev/maxScale"        = "1000"  
                "run.googleapis.com/cloudsql-instances"   = google\_sql\_database\_instance.todo\_database.connection\_name  
                "run.googleapis.com/client-name"          = "terraform"  
                "run.googleapis.com/vpc-access-egress"    = "all"  
                "run.googleapis.com/vpc-access-connector" = google\_vpc\_access\_connector.connector.id  
            }  
        }  
    }  
    autogenerate\_revision\_name = true  
    depends\_on = [  
        null\_resource.cloudbuild\_api,  
        google\_project\_iam\_member.secretmanager\_secretAccessor  
    ]  
}

**Open Cloud Run API Service to be world readable**

This API layer of the application will be called by the user's browser, but by default Cloud Run services are not public. In order for users to consume this service, we have to open permissions on these services to be accessible to the world.

resource "google\_cloud\_run\_service\_iam\_policy" "noauth\_api" {  
    location    = google\_cloud\_run\_service.api.location  
    project     = google\_cloud\_run\_service.api.project  
    service     = google\_cloud\_run\_service.api.name  
    policy\_data = data.google\_iam\_policy.noauth.policy\_data  
}

**Create Artifact from Front End**

resource "null\_resource" "cloudbuild\_fe" {  
    provisioner "local-exec" {  
        working\_dir = "${path.module}/code/frontend"  
        command     = "gcloud builds submit . --substitutions=\_REGION=${var.region},\_BASENAME=${var.basename}"  
    }  
    depends\_on = [  
        google\_artifact\_registry\_repository.todo\_app,  
        google\_cloud\_run\_service.api  
    ]  
}

**Deploy Frontend container to cloud run**

resource "google\_cloud\_run\_service" "fe" {  
    name     = "${var.basename}-fe"  
    location = var.region  
    project  = var.project\_id  
    template {  
        spec {  
            containers {  
                image = "${var.region}-docker.pkg.dev/${var.project\_id}/${var.basename}-app/fe"  
                ports {  
                    container\_port = 80  
                }  
            }  
        }  
    }  
    depends\_on = [null\_resource.cloudbuild\_fe]  
}

**Open Cloud Run front end service to be world readable**

resource "google\_cloud\_run\_service\_iam\_policy" "noauth\_fe" {  
    location    = google\_cloud\_run\_service.fe.location  
    project     = google\_cloud\_run\_service.fe.project  
    service     = google\_cloud\_run\_service.fe.name  
    policy\_data = data.google\_iam\_policy.noauth.policy\_data

./code/database/load\_schema.sh

PROJECT=$1  
SQLNAME=$2  
  
SQLSERVICEACCOUNT=$(gcloud sql instances describe $SQLNAME --format="value(serviceAccountEmailAddress)" | xargs)  
gsutil mb gs://$PROJECT-temp   
gsutil cp schema.sql gs://$PROJECT-temp/schema.sql  
gsutil iam ch serviceAccount:$SQLSERVICEACCOUNT:objectViewer gs://$PROJECT-temp/  
gcloud sql import sql $SQLNAME gs://$PROJECT-temp/schema.sql -q  
gsutil rm gs://$PROJECT-temp/schema.sql  
gsutil rb gs://$PROJECT-temp

./code/middleware/clouldbuild.yaml

**This code makes a Docker image for the middleware layer.**

name: 'gcr.io/cloud-builders/docker'  
args: [ 'build', '-t', '$\_REGION-docker.pkg.dev/$PROJECT\_ID/$\_BASENAME-app/api', '.' ]  
  ```  
#### Push API container to Artifact Registry  
Pushing the container to Artifact Registry makes it possible for Cloud Run to   
get the image and serve it.  
  
``` yaml  
name: 'gcr.io/cloud-builders/docker'  
args: ['push', '$\_REGION-docker.pkg.dev/$PROJECT\_ID/$\_BASENAME-app/api']

substitutions:  
  \_REGION: us-central1  
  \_BASENAME: todo

./code/frontend/clouldbuild.yaml

The front end is completely static HTML/JS/CSS. The app needs to point to the URL for the API service we just created, but Cloud Run services are assigned a url with a randomized string. This 'massage script' captures that randomized URL and injects it in to the code of the static JS in this container.

name: 'gcr.io/google.com/cloudsdktool/cloud-sdk'  
entrypoint: bash  
args: [ './massage.sh', '$\_REGION' ]

Build API Container

name: 'gcr.io/cloud-builders/docker'  
args: [ 'build', '-t', '$\_REGION-docker.pkg.dev/$PROJECT\_ID/$\_BASENAME-app/fe', '.' ]

Push API Container

name: 'gcr.io/cloud-builders/docker'  
args: ['push', '$\_REGION-docker.pkg.dev/$PROJECT\_ID/$\_BASENAME-app/fe']

substitutions:  
  \_REGION: us-central1  
  \_BASENAME: todo

./code/frontend/massage.sh

API=$(gcloud run services describe todo-api --region=$1 --format="value(status.url)")  
stripped=$(echo ${API/https:\/\//})  
sed -i"" -e "s/127.0.0.1:9000/$stripped/" www/js/main.js

This above command injects the endpoint for the Middleware into the JavaScript of the front end.

Conclusion:

We now have a simple 3 tier to do application running on Cloud Run in your project