CERT

Sunday, October 2, 2022 9:31 AM

Needed stuffs to master:
Gcloud sdk for everything
Service like cloud run,functions,gce,gke,gcs,iam,deployment manager,app engine

TERRAFORM

Monday, October 3, 2022 12:49 PM

1) Scope(confirm the resources required for a project)
2) Author(author the config files based on the4 scope)
3) Initialize(download the provider plugins and initialie directory)
4) Plan(view execution plan for resources created,modified,destroyed)
5) Apply(create actual infrastructure resources)

Versions available:
- Terraform open source(only CLI,no concurrent deployments)
- Terraform Cloud(CLI,GUI,concurrent deployments)
- Terraform enterprise(GUI,CLI, conurrent deployments,secure deployments)

Authentication :
- Using gcloud sdk
- Using service account in vm's
- Using env vars for service account keys in onprem

Example workflow:
- Create .tf files
  Resource "google_compute_network" "my_network"{
  Name="my_vpc"
  }
- Terraform init
- Terraform plan
- Terraform apply
- Terraform destroy
- Terraform fmt(autoformat to match canonical conventions)

Need to organize .tf files in terms of directories and files for each functions like main.tf, providers.tf, variables.tf, out puts.tf

In the main.tf the actual resource configs are written

In the providers.tf the cloud provider plugin is downloaded and given configuration options like project id ,etc..

In the variables.tf we define the variables that looks into a terraform.tfvars for runtime stuff like
Variable location{
..
} in variables.tf and
Localtion=''US'' in terraform.tfvars

In the outputs.tf we can define outputs in order to retrieve the created resources config url or similar to bucket url
Output "buckety_URL" {
    Value = google_storage_bucket.mybucket.URL
}

In the terraform.tfstate stores the state for the resources we create using terraform locally,remotely

Modules:
- Is a set of terraform config files in a single directory

For running the terraform validator we use gcloud beta terraform vet

When developing code we tend to use the resource attribute's values
For that we can use **resourcename.resourcetype.attribute**

Meta-arguments:
- To customize the behaviour of resources

1) Count -> create multiple instances
   {
   Count=3
   Name = "devVM${count.index + 1}"
   }
2) For_each -> create multiple resource instances as per a set of strings
   {
   For_each = toset(["us-central1-a","asia-east-b","europe-westr-a"]}
   Name = "dev-${ech.value}"

Zone = each.value
}

3) Depends_on -> specify explicit dependency
4) Lifecycle -> define lifecycle of a resource

Dependencies:

- Implicit dependency(dependencies known to terraform are detected automatically)
  When we give the **resourcename.resourcetype.attribute** inside the resource definition to take the value from other resource defined in the same file
- Explicit dependencies should be configured explictly and are not known by default)
  Depends_on = [google_compute_instances.server]

Variables:

- Should specify in variables.tf
  Variable "name" {
  Type= string
  Description = "noting"
  Default = "US"
  Sensitive = true # this will not be shown in the terraform plam,apply commands ops
  }
- in the main.tf we can reference it by "${var.instance_name}"
- Also we can alternatively pass variables in .tfvars file and pass it through (tf apply -var-file myvars.tfvars)
- Tf apply -var project_id="projectid"
- TF_VAR_project_id="projectid" tf apply
- Tf apply (if using terraform.tfvars)
- In the terraform.tfvars we can specify variables as a line (key=val)
- If no value is specified then terraform asks when applying

Validation{
Conditoin = containsn(["","",""],var.varname)
Error_message = "Ddddd"
}
(

```
variable "location" {
  description = "The Azure Region in which all resources in this example should be provisioned"
 validation{
condition = contains(["ram","shankar","is"],var.location)
error_message = "Ddddd"
}
}
```

)

- For validation

Outputs:

- Usually in outputs.tf file

Output "picture_URL" {
Description: "something"
Value = google_storage_bucket.picture.self_link
Sensitive=true
}

Terraform registry:

- Interactive resource for discovering providers,modules
- Solutions developed by hashicorp,thirdparty etc..

Cloud foundation kit:

- Provides a series of reference modules for terraform that reflect gcp best pratices
- Modules in this can be used without modification to quickly build a repeatable enterprise ready foundation in gcp

terraform graph | dot -Tsvg > graph.svg(to view dependency graph)

Modules:

Module <name> {
Source = "./dir" (can be terraform registry,github,bitbucket,http urls,gcs bucket)
}
Also we can give the module a attribute given at runtime like
Module "" {
Source = ""
Network_name = module.my_network.network_name
}

scrip

Terraform states:
We can add a backend.tf file to store the state file remotely instead of locally

Terraform {
Backend "gcs" {
Bucket = "name"
Prefix = "terraform/state"
}}
terraform init -migrate-state
terraform refresh

terraform show - to show the state

terraform plan -out static_ip (to save the plan for future use the same)
Terraform apply "static_ip" (for using the same plan outputted)

terraform taint google_compute_instance.vm_instance (to tell terraform to recreate the instance)
terraform taint module.instances.google_compute_instance.tf-instance-461793

```
resource "google_compute_instance" "vm_instance" {
  name         = "terraform-instance"
  machine_type = "f1-micro"
  tags         = ["web", "dev"]
  provisioner "local-exec" {
    command = "echo ${google_compute_instance.vm_instance.name}:
${google_compute_instance.vm_instance.network_interface[0].access_config[0].nat_ip} >> ip_address.txt"
  }
  # ...
}
```
(we can use provisioners like this to exec some commands)

In the module directory we will have the .tf files with variables (parameterized)
When we call the modules with module{} block we specify some attributes inside this block thjose are variables used in the modules

After defining module in the main.tf file we have to run terraform import module.modulename.resourcetype.resourcename to import the existing resources that are already available in gcp

Like 2 instances already available should be declared inside a directory/module and in the main tf file initialize the module with source as the director and run the command accordingly

PRACTICALS

Tuesday, October 4, 2022 8:49 PM

Restart/Replace VM's:

- When we replace VMs in instance-group
  We can temporarily add additional instances like maxSurge(How many new instances can be temporarily added at the same time while updating),we can add max unavailable instances (maxUnavailable(Highest number or percent of instances that can be unavailable at the same time during an update))
- When we restart only the manunavilable will be showed

Update VM's:

- When we update VM'S in MIG's update type is Selective(Update VMs in this group when they are replaced, refreshed, or restarted, except during auto-healing. (API name: OPPORTUNISTIC)),Automatic(Start proactive updates automatically. (API name: PROACTIVE))
- For automatic we have the options same as the restart/replace instances options
- We can do a canary testing like use out of 5 vm's 2 one instance template and 3 to a new insatcne template or 100% to another template

CLOUD FUNCTIONS:

- Trigger types available in cloud functions gen1 are HTTP,pubsub,gcs,firestore
- Trigger types available in gcf gen2 are endless through eventarc triggers that has most of the google apis and services and its event triggers
- Based on the request load instances are created to handle the load we can also modify the autoscaling instances in the runtime section
- The memory allocated for the gcf instance can be upto 8 gb In gen1 and upto 32 gb in gen 2
- By default, your function can send requests to the internet, but not to resources in VPC networks. To send requests to resources in your VPC network, create or select a VPC connector already created in the same region as the function.(Serverless VPC Access API)
- Serverless VPC Access allows Cloud Functions, Cloud Run (fully managed) services and App Engine standard environment apps to access resources in a VPC network using the internal IP addresses of those resources

1st gen	2nd gen
Concurrency 1 request	Up to 1000 requests*
Event Sources 8 trigger types	90+ Eventarc sources
Execution Time 9 min max	60 min max (HTTP only)
Traffic Management Not supported	Ability to split traffic and roll back to prior revision*(for cloud run)

- When you deploy your function's source code to Cloud Functions, that source is stored in a Cloud Storage bucket. Cloud Build then automatically builds your code into a container image and pushes that image to a image registry (either Container Registry or Artifact Registry)

CLOUD RUN:

- Creating a service in cloud run means creating a service in kubernetes world as it generates a yaml file after creating a service
- In the containers,connections,security section it is mostly same as the gcf but here in the connections sql connection is there and also **use http/2 endtoend** checkbox for if we use Grpc streaming server or it can handle request by itself

- Execution environment is picked automatically bhy cloud run if given default else wer need to select default/firstgen/secondgen(full linux compatibility(preview))
- A **Cloud Run job** executes containers to completion. Job name and region cannot be changed later.

Metadata in GCE:

- All the vm's inherit the metadata server values by default without any additional authorization
- We can also query the metadata server using urls
- Root url : http://metadata.google.internal/computeMetadata/v1
- Header: Metadata-Flavor: Google or X-Google-Metadata-Request: True
- Any requests that contain the header X-Forwarded-For are automatically rejected by the metadata server. This header generally indicates that the request was proxied and might not be a request made by an authorized user. For security reasons, all such requests are rejected.
- Project metadata is stored under http://metadata.google.internal/computeMetadata/v1/project/project-id
- Using the same url with project/attributes/ we have several atttributes to disable or enable oslogin,windowsssh,
- SSH keys managed by OS Login aren't visible in metadata.
- Instance metadata is stored under : http://metadata.google.internal/computeMetadata/v1/instance/
  Paths include disks/(device-name,index,interface.mdoe,type),hostname,id,image,machine-type,name,network-intefaces/(accessconfigs/externalip,gateway,ip,network,serviceaccount/(),etc..)
- When we want to see the directory listings available and query according to that we need to query like
- curl "http://metadata.google.internal/computeMetadata/v1/instance/network-interfaces/0/access-configs/" -H "Metadata-Flavor: Google"
  Result will be like 0/ and we need to use this in the url to further dive into like access-configs/0/external-ip
- Adding a ?recursive=true in the url end will give us all the directory listings inside the directory
- ?wait_for_change=true will be like --watch in kubernetes
- For custom metadata that we want to add for boolean values YES Y,1,Yes also can be added as the value
- If we apply metadata to the metadata page it applies to all the vm instances and if we want we can them individually

Creating instance gce:

- We can manually add our own ssh keys in security tab
- In the management section we can give the strtuip script or metadata and also enable deletion protection like cloud sql
- Also for vm provisioning model standard(normal),spot(preemptible), when on host maintenance is occuring we can either migrate vm or terminate vm

Nested virtualization:

- For enabling it we use --enable-nested-virtualization when craeting instances usiong gcloud command
- Or by exporting the vm using gcloud command to a yaml filepath and updating into that like advancedMachineFeatures:
  enableNestedVirtualization: true
  And update the vm form file
- Or by creating an image from a disk and specify --licenses "https://www.googleapis.com/compute/v1/projects/vm-options/global/licenses/enable-vmx" when creating image using gcloud

- To verify grep -cw vmx /proc/cpuinfo
- Creating a nested vm is called as L2(level2) VM that is created from L1 vm that is vm with nested vm enabled

OS LOGIN:

- OS Login simplifies SSH access management by linking your Linux user account to your Google identity. Administrators can easily manage access to instances at either an instance or project level by setting IAM permission
- Enabling oslogin through metadat key pair like enable-oslogin=TRUE or while creating vm manage access via iam roles
- roles/compute.osLogin,roles/compute.osAdminLogin are the roles required for the iam user for connecting,compute.projects.get access for ssh access
- For users that are outside of your organization to access your VMs, in addition to granting an instance access role, grant the roles/compute.osLoginExternalUser role, which enables POSIX account creation. This role must be granted at the organization level by an organization administrator
- enable-oslogin-2fa=TRUE for 2fa
  gcloud compute os-login ssh-keys add \
  --key-file=KEY_FILE_PATH \
  --project=PROJECT \
  --ttl=EXPIRE_TIME
  For adding ssh keys that use oslogin

For transfering files use - gcloud compute scp LOCAL_FILE_PATH VM_NAME:~
gcloud compute scp --recurse VM_NAME:REMOTE_DIR LOCAL_DIR
scp -i ~/.ssh/my-ssh-key LOCAL_FILE_PATH USERNAME@IP_ADDRESS:~
scp -i ~/.ssh/my-ssh-key USERNAME@IP_ADDRESS:REMOTE_FILE_PATH LOCAL_FILE_PATH

Migrate for compute engine/virtual machines:

- In the onprem migrate connector is installed and the connector connects to google apis though port 443
- For running your migrated vm's we can have them in the host project or add additional target projects for running those as well in them
- Steps:
  - you must create a vCenter user account (onprem) with the permissions required by the Migrate Connector to access your vSphere environment
  - Create ssh key pair
  - On gcp create a user account for registration process(for connector) and service account for the migrate connector for migration
  - Configuring firewall rules for the migration to occur

We can create vm's as a bulk using gcloud
gcloud compute instances bulk create \
(--name-pattern="NAME_PATTERN" | --predefined-names=[PREDEFINED_NAMES]) \
--region=REGION \
--count=COUNT \
[--min-count=MIN_COUNT \ ]
[--location-policy=LOCATION_POLICY \ ] (us-east1-c=allow,us-central1-c=deny)
[--target-shape=TARGET_SHAPE](any_single_zone,balanaced,any)

Sole tenancy VM's:

- We need to first create node template that appears to all the nodes in a node group
- In the node template we give a node type which will be in the format ((machinetype/(compute,memory optimized))-node-cpu-memory) , optionally localssds,gpu accelerators,node affinity labels
- Configure autoscaling

- What to do when maintenance(restart all/migrate within node groups)
- We can either share the node groups created with projects or not at all

Google supports virtual display devices on Linux instances and on Windows instances that use any x64-based Windows images v20190312 or later. That can be enabled when creating vm or after stopping vm's

Leap seconds in Unix time are commonly implemented by repeating the last second of the day, to keeping server time in sync, NTP(network time protocol) is helpful in the rare case of a leap second chronyc sources - inside vm to see the status of ntp server

Virtio RNG is a paravirtualized random number generator, to check lsmod | grep rng , cat /dev/random | rngtest -c 1000

For higher bandwidth vm's when creating vm mention only within N2, N2D, C2, or C2D VM and in boot section gVNIC-compatible or custom images and in the networking seciton choose a gVNIC network interface(gVNIC is required to support higher network bandwidths such as the 50-100 Gbps speeds that can be used for distributed workloads on VMs that have attached GPUs. Also, gVNIC is required when working with some VM shapes that are meant for optimal performance when using VMs)

PRACTICALS 2

Thursday, October 6, 2022 4:17 PM

Gcloud cloud-shell gcp/mount

In the google kubernetes engine when creating we can select the control plane version if it needs to be static(manual upgrade(need to specify manually whether to use autorepair/upgrades)) or a release version(automatic node repair autoupgrade)

We can also enable spot vm's for nodes while creating cluster to minimize cost

In autopilot configuration it does all the autoscaling itself but in standard mode if we want to autoscae then we can specify autoscaling by giving min nodes and max nodes

We can also enable vertical pod autoscaling in standard mode

Node auto-provisioning automatically manages a set of node pools on the user's behalf. Without node auto-provisioning, GKE considers starting new nodes only from the set of user created node pools. With node auto-provisioning, new node pools can be created and deleted automatically.

Enable control plane authorized networks to block untrusted non-GCP source IPs from accessing the Kubernetes control plane through HTTPS.

APP ENGINE:

- Two runtimes are there firstgen.second gen
- Fo each generation runtimes thee are multiple instance classes that define the memory limit of each instance generally second gen is double than first gen in terms of limit(memory,cpu)
- When deploying app we can specify our own user managed service account(gcloud .. --service-account,app.yaml: service_account:value) else default app engine sa will be created
- The service gent account will be in the format service-PROJECT_NUMBER@gcp-gae-service.iam.gserviceaccount.com so even if we delete the service account by mistake we can add this to iam(App Engine Standard environment Service Agent role.)
- We can communicate to services and versions inside of the app by using http request for below It should not also exceed the 63 characters or it will throw dns error https://VERSION-dot-SERVICE-dot-PROJECT_ID.REGION_ID.r.appspot.com
- In the app.yaml file we can specify **max_concurrent_requests** in order to mnage the concurrent request a single instance can receive
- All HTTP/2 requests will be translated into HTTP/1.1 requests when forwarded to the application server.
- default_expiration: "4d 5h" in app.yaml specifies to expire the static files duration after the first request
- Env_variables,error_handlers,handlers,instance_class are available in app.yaml config
- In handlersw we can define for each url what static file should run
- If using instance class F1 or higher we can set automatic scaling with attributes like max/min_instances,max/min_idle_instances ,max/min_pending_latency,target_cpu/throughput_utilization
- We can create an index.yaml file for apps that request datastore (gcloud app deploy index.yaml)
- For java based apps use web.xml where we give servelet,servelte-mapping for app and deploy the file
- For php apps php.ini is there
- For requests from a Shared VPC, traffic is only considered internal if the App Engine app is deployed in the Shared VPC host project. If the App Engine app is deployed in a Shared VPC service project, only traffic from networks owned by the app's own project is internal. All other traffic, including traffic from other Shared VPCs, is external.
- When deploying app we can specify --ingress controls and values as all, internal-only, or internal-and-cloud-load-balancing and for egress as private-ranges-only,all-traffic
- paths with /_ah/ are not blocked in the flexible environment

After deploying the app we can deploy this dispatch.yaml file containgin this to service services based on urls by (gcloud app deploy dispatch.yaml)

```
dispatch:
  # Send all mobile traffic to the mobile frontend.
  - url: "*/mobile/*"
    service: mobile-frontend

  # Send all work to the one static backend.
  - url: "*/work/*"
    service: static-backend
```

Basic scaling:

```
basic_scaling:
  max_instances: 11
  idle_timeout: 10m
```

Manual scaling:

```
manual_scaling:
  instances: 5
```

Cron.yaml (gcloud app deploy cron.yaml)

```
cron:
- description: "test dispatch vs target"
  url: /tasks/hello_service2
  schedule: every 1 mins
  target: service1
```

For flex:

```
runtime: python
env: flex
entrypoint: gunicorn -b :$PORT main:app

runtime_config:
  python_version: 3

manual_scaling:
  instances: 1
resources:
  cpu: 1
  memory_gb: 0.5
  disk_size_gb: 10
```

DEPLOYMENT MANAGER:

- gcloud deployment-manager types list(for available resource types)
- Also we can create a ninja template and pass that in resource type to automatically create
- For previewing we can use gcloud deployment-manager deployments create example-config --config configuration-file.yaml --preview
- Cancel the preview if you think it is not working - gcloud deployment-manager deployments cancel-preview example-config
- We can define outputs like

```
...
outputs:
- name: databaseIp
  value: $(ref.my-first-vm.networkInterfaces[0].accessConfigs[0].natIP)
- name: machineType
  value: {{ properties['machineType'] }}
- name: databasePort
  value: 88
```

10.0.0.0/7 is for 10.0.0. addressses(limit in subnet)
192.168.0.0/13 for 192.168.* addresses
192.0.0.0/2 for 192.* addresses
10.128.0.0/16 is the limit for automode subnets

In BYOP section in networks Public Advertised Prefix(PAP) is the cidr range which we own and validate that for ringing into gcp

To set acl

gsutil acl ch -u USER_EMAIL:PERMISSION gs://BUCKET_NAME
gsutil acl set JSON_FILE gs://BUCKET_NAME (JSON_FILE contains the permissions for acl)
gsutil cp -a bucket-owner-read paris.jpg gs://example-travel-maps (this is a predefined permission)

gsutil defacl ch -u jane@gmail.com:READER gs://example-travel-maps

An HMAC key is a type of credential and can be associated with a service account or a user account in Cloud Storage. You use an HMAC key to create signatures which are then included in requests to Cloud Storage HMAC keys have two primary pieces, an access ID and a secret.

You can have a maximum of 5 HMAC keys per service account. Deleted keys do not count towards this limit.
gsutil hmac create SERVICE_ACCOUNT_EMAIL (it will rewturn the accessid,secret)
gsutil hmac list
gsutil hmac get KEY_ACCESS_ID
gsutil hmac delete ACCESS_KEY_ID

gsutil signurl -d 10m Desktop/private-key.json gs://example-bucket/cat.jpeg
gsutil signurl -m PUT -d 1h -c CONTENT_TYPE -u gs://BUCKET_NAME/OBJECT_NAME