Documentação da infraestrutura Terraform

Este documento descreve os arquivos presentes na pasta `terraform/` do projeto "mensagem". Ele explica o propósito de cada arquivo e as principais entradas/saídas e recursos gerenciados.

Sumário

Observação: este repositório também contém a aplicação (FastAPI), Dockerfile e docker-compose; aqui documentamos apenas os arquivos de infraestrutura em `terraform/`.

---

alb.tf

Propósito:

Cria o Application Load Balancer (ALB), o target group e o listener HTTP (porta 80) que encaminha para o target group.

Principais recursos:

- aws_lb.app: o ALB público
- aws_lb_target_group.app: target group que aponta para as tasks do ECS (usa `var.container_port`)
- aws_lb_listener.http: listener na porta 80 com ação default para encaminhar ao target group

Notas:

- Existe código comentado para um listener HTTPS caso queira usar ACM (`certificate.tf`).
- Health check configurado para `/` (matcher `200-399`). Ajuste se sua aplicação expor health em outro caminho.

---

certificate.tf

Propósito:

Recurso opcional para criar/usar um certificado ACM (muito provavelmente comentado por padrão).

Uso esperado:

- Criar ou referenciar um certificado ACM válido para usar com `aws_lb_listener` HTTPS.
---
ecr.tf
Propósito:
Cria o repositório AWS ECR que armazenará a imagem Docker da aplicação.
Recursos:
- aws_ecr_repository.app
Notas:
- Usado por pipelines CI/CD ou deploy manual para fazer push de imagens e referenciá-las n
a task definition.
---
ecs_cluster.tf
Propósito:
Cria o cluster ECS onde as tasks/serviços serão executados.
Recursos:
- aws_ecs_cluster.app
---
ecs_service.tf
Propósito:
Define o `aws_ecs_service` que roda as tasks no Fargate, faz o registro no target group do
ALB e configura a rede.
Campos chave:
- `cluster`: referência ao cluster (aws_ecs_cluster.app.id)
- `task_definition`: referência à `aws_ecs_task_definition.app.arn`
- `desired_count`: número de instâncias desejadas
- `launch_type = "FARGATE"`
- `load_balancer`: bloco que conecta o serviço ao `aws_lb_target_group.app`
- `network_configuration`: subnets privadas e security groups (aws_security_group.ecs_task
s)
- `depends_on = [aws_lb_listener.http]`: força a criação do listener antes do service (evi
ta condição de corrida na primeira aplicação)
Notas:
- Confirme que as `aws_subnet.private` existem e são as corretas para tarefas Fargate.
- Se usar autoscaling/prod, considere remover `desired_count` fixo e usar `aws_appautoscal
ing_target`.
---
ecs_task.tf
Propósito:
Define a `aws_ecs_task_definition` que descreve o container (imagem ECR, portas, logs, var
iáveis de ambiente e secrets).
Pontos importantes:
- `requires_compatibilities = ["FARGATE"]`, `network_mode = "awsvpc"`
- `execution_role_arn` e `task_role_arn` apontam para `aws_iam_role.ecs_task_execution_rol

e`
- `container_definitions` use `jsonencode` para declarar:
  - image: `${aws_ecr_repository.app.repository_url}:${var.ecr_image_tag}`
  - portMappings: `containerPort = var.container_port`
  - logConfiguration: usa `awslogs` para enviar logs ao CloudWatch (grupo `aws_cloudwatch_
log_group.app`)
  - environment / secrets: exemplo com `ENV=production` e `APP_SECRET` vindo do Secrets Ma
nager
---
iam.tf
Propósito:
Define papéis (IAM roles) e políticas necessárias para o ECS Task Execution (ex.: acesso a
 ECR, CloudWatch Logs, SecretsManager).
Recursos típicos:
- aws_iam_role.ecs_task_execution_role
- aws_iam_role_policy_attachment para anexar `AmazonECSTaskExecutionRolePolicy`, `AmazonEC
2ContainerRegistryReadOnly`, etc.
Observação:
- Revise as permissões em `iam.tf` antes de colocar em produção e aplique o princípio do m
enor privilégio.
---
logs.tf
Propósito:
Cria o grupo de logs no CloudWatch para a aplicação.
Recursos:
- aws_cloudwatch_log_group.app
Notas:
- O nome do grupo é usado na `task_definition` (field `awslogs-group`).
- Ajuste `retention_in_days` conforme necessidade de retenção de logs.
---
outputs.tf
Propósito:
Define saídas (outputs) úteis após o `terraform apply`, como endereço do ALB, ARN do ECR,
etc.
Uso:
- Facilita integrar com scripts/pipelines que leem os valores produzidos pelo Terraform.
---
provider.tf
Propósito:
Configura o provider AWS (região, profile, versão do provider) e quaisquer backends/state.
Notas:
- Se houver bloco `backend` (S3), revise credenciais e permissões.
- Para validação local rápida, você pode rodar `terraform init -backend=false`.

---
secrets.tf
Propósito:
Cria recursos no AWS Secrets Manager para armazenar segredos (por exemplo `APP_SECRET`).
Recursos:
- aws_secretsmanager_secret.app
- aws_secretsmanager_secret_version
Notas:
- Verifique quem tem acesso a esses segredos e como seus segredos são injetados nas tasks (via `secrets` no `container_definitions`).
---
sg.tf
Propósito:
Define security groups para o ALB e para as tasks ECS.
Recursos principais:
- aws_security_group.alb: permite inbound 80 e 443 (0.0.0.0/0)
- aws_security_group.ecs_tasks: permite inbound na porta da aplicação apenas do security g roup do ALB
Notas:
- Essa é uma configuração comum: ALB público + tasks em subnets privadas acessíveis soment e pelo SG do ALB.
---
terraform.tfvars.example
Propósito:
Arquivo de exemplo com valores de variáveis que o usuário deve copiar para `terraform.tfva rs` e ajustar (p.ex. `project_name`, `aws_region`, `container_port`, `desired_count`, `ecr _image_tag`).
---
variables.tf
Propósito:
Lista e descreve variáveis usadas pelo conjunto de arquivos Terraform (nomes, tipos, valor es padrão e descrições).
Importante:
- Revise as variáveis e documente valores sensíveis em um `terraform.tfvars` fora do repos itório ou via CI/CD segredos.
---
vpc.tf
Propósito:
Cria a VPC, subnets (públicas/privadas), route tables, IGW, NAT gateway(s) e outras peças de rede necessárias para rodar o ALB (em public subnets) e as tasks ECS (em private subnet s).
Notas:
- Certifique-se de que as subnets privadas têm rotas para o NAT gateway para permitir que

tasks alcancem ECR/SSM/SecretsManager se necessário.

Recomendações gerais e checklist antes de aplicar em produção

- Validar com `terraform validate` e `terraform plan`.
- Manter o state remoto (S3 + DynamoDB) para colaboração e prevenção de conflitos.
- Usar versões fixas de providers e módulos quando possível.
- Revisar IAM roles/policies e aplicar princípio do menor privilégio.
- Testar a aplicação localmente via Docker antes de enviar imagens ao ECR.

Se quiser, eu posso:

- Gerar um PDF desta documentação automaticamente e deixá-lo em `docs/infra_documentation.pdf` (posso criar aqui se você autorizar a execução de um pequeno script e instalação de dependência Python `reportlab`).
- Rodar `terraform init -backend=false` e `terraform validate` e trazer o resultado para você.