

Secret Management - Adding new secrets to an ECS service

Overview

Services within an application sometimes need to connect to database or external services. This requires a password or secret key in order to access the database or external services. Storing a secret within the source code is a terrible idea. The way to solve this problem to provide these secrets during infra service provisioning.

Content

The component [fargate-service](#) provisions a docker container into an ECS(AWS Elastic Container Service) cluster with networking, service discovery registration, and health checks provided. It also provides settings on how to run and configure the container (e.g., private/public, inject secrets, CPU, and memory). As part of service provisioning, the module also creates a Secret Manager which is used to store service secrets with encryption, using the KMS key.

This component gives the ability to create 1 or multiple Secrets Manager secrets and have their values automatically injected into the service container at startup. This is done by way of environment variables defined in the service's ECS Task Definition and pointing to their respective Secrets Manager secret.

Variables are used to make this work:

- **secrets:** Map which contains `secret_name` in AWS Secrets Manager and `secret_json` which is the value containing key/value pair that will become the name and value of the environment variables injected into the container at startup.

Each item in the `secrets` map will become a Secrets Manager secret whose name is provided by the map `key` and whose value will be provided by the `value` of the map. The map `value` is a JSON-encoded string that will contain a map of all values for that secret.

At runtime, the container will be injected with environment variables named after the `name` field and whose value will be the content of the `value` (or similarly the JSON-encoded version of the corresponding Secrets Manager secret value).

For example in a [canary-spoke example](#) blueprint, the `service-api` service

```
secrets = [{
  name   = "default"
  value  = "${jsonencode(var.api_secret_value)}"
}]
```

Above terraform snippet creates new secret manager as `default` with secret key/values from the variable `api_secret_value`.

i The variable `api_secret_value` can be configured in the terraform workspace as variable. Below screenshot shows an variable application_name with its value. These values can be masked to hide.

min-au-infra / Workspaces / 01-paspo-kedro-base / Variables

01-paspo-kedro-base ⓘ

Runs States Variables

Variables

These variables are used for all plans and applies in this workspace. Workspaces using Terraform 0.10.0 or later can also load default values from any `*.auto.tfvars` files in the configuration.

Sensitive variables are hidden from view in the UI and API, and can't be edited. (To change a sensitive variable, delete and replace it.) Sensitive variables can still appear in Terraform logs if your configuration is designed to output them.


When setting many variables at once, the [Terraform Cloud Provider](#) or the [variables API](#) can often save time.

Terraform Variables

These [Terraform variables](#) are set using a `terraform.tfvars` file. To use interpolation or set a non-string value for a variable, click its HCL checkbox.

Key	Value
application_name Name of the application, used in naming AWS ECR Repos, Gitlab Group and Gitlab Project	paspo

The below terraform code is taken from the [module](#), it creates a task execution role which allows ECS service to execute the tasks.

 More details about ECS task execution role https://docs.aws.amazon.com/AmazonECS/latest/developerguide/task_execution_IAM_role.html

```
resource "aws_iam_role" "task_execution" {
  name           = "${var.service_name}-ecs-task-execution"
  assume_role_policy = data.aws_iam_policy_document.
task_execution_assume_role_policy.json

  tags = merge({
    Name           = "${var.service_name}-task-execution"
    ChargeCode     = var.charge_code
  }, var.custom_tags)

  provider = aws.service
}

data "aws_iam_policy_document" "task_execution_assume_role_policy" {
  statement {
    sid      = ""
    effect   = "Allow"
    actions  = ["sts:AssumeRole"]

    principals {
      type       = "Service"
      identifiers = ["ecs-tasks.amazonaws.com"]
    }
  }

  provider = aws.service
}

resource "aws_iam_role_policy_attachment" "task_execution" {
  role           = aws_iam_role.task_execution.id
  policy_arn     = "arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy"

  provider = aws.service
}
```

Once this task execution role created and attached to ECS service, the service will have all the necessary access to execute tasks from the task definition. But the service still does not have privileges to access secrets defined in the secret manager. In order to allow ECS service to access secret manager, the execution role has to be updated with a policy that allows the secret manager access. Below terraform snippet from the [module](#) highlights role being updated with a new policy

```

resource "aws_iam_role_policy" "task_execution_secrets" {
  for_each = var.secrets
  name      = "${var.service_name}-ecs-task-execution-secrets"
  role      = aws_iam_role.task_execution.id

  policy = <<EOF
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["secretsmanager:GetSecretValue"],
      "Resource": ${jsonencode(local.secret_manager_arns)}
    }
  ]
}
EOF

  provider = aws.service
}

```

Relevant materials

- Fargate service hosted module <https://app.terraform.io/app/min-au-infra/modules/view/fargate-service/aws/6.0.4>
- Source for Fargate service <https://gitlab.com/mc-components/terraform-aws-fargate-service>
- More about ECS task execution policy https://docs.aws.amazon.com/AmazonECS/latest/developerguide/task_execution_IAM_role.html