Vault Provider

The Vault provider allows Terraform to read from, write to, and configure Hashicorp Vault (https://vaultproject.io/).

> **Important** Interacting with Vault from Terraform causes any secrets that you read and write to be persisted in both Terraform's state file *and* in any generated plan files. For any Terraform module that reads or writes Vault secrets, these files should be treated as sensitive and protected accordingly.

This provider serves two pretty-distinct use-cases, which each have their own security trade-offs and caveats that are covered in the sections that follow. Consider these carefully before using this provider within your Terraform configuration.

Configuring and Populating Vault

Terraform can be used by the Vault adminstrators to configure Vault and populate it with secrets. In this case, the state and any plans associated with the configuration must be stored and communicated with care, since they will contain in cleartext any values that were written into Vault.

Currently Terraform has no mechanism to redact or protect secrets that are provided via configuration, so teams choosing to use Terraform for populating Vault secrets should pay careful attention to the notes on each resource's documentation page about how any secrets are persisted to the state and consider carefully whether such usage is compatible with their security policies.

Except as otherwise noted, the resources that write secrets into Vault are designed such that they require only the *create* and *update* capabilities on the relevant resources, so that distinct tokens can be used for reading vs. writing and thus limit the exposure of a compromised token.

Using Vault credentials in Terraform configuration

Most Terraform providers require credentials to interact with a third-party service that they wrap. This provider allows such credentials to be obtained from Vault, which means that operators or systems running Terraform need only access to a suitably-privileged Vault token in order to temporarily lease the credentials for other providers.

Currently Terraform has no mechanism to redact or protect secrets that are returned via data sources, so secrets read via this provider will be persisted into the Terraform state, into any plan files, and in some cases in the console output produced while planning and applying. These artifacts must therefore all be protected accordingly.

To reduce the exposure of such secrets, the provider requests a Vault token with a relatively-short TTL (20 minutes, by default) which in turn means that where possible Vault will revoke any issued credentials after that time, but in particular it is unable to retract any static secrets such as those stored in Vault's "generic" secret backend.

The requested token TTL can be controlled by the `max_lease_ttl_seconds` provider argument described below. It is important to consider that Terraform reads from data sources during the `plan` phase and writes the result into the plan. Thus a subsequent `apply` will likely fail if it is run after the intermediate token has expired, due to the revocation of the secrets that are stored in the plan.

Except as otherwise noted, the resources that read secrets from Vault are designed such that they require only the *read* capability on the relevant resources.

Provider Arguments

The provider configuration block accepts the following arguments. In most cases it is recommended to set them via the indicated environment variables in order to keep credential information out of the configuration.

- `address` - (Required) Origin URL of the Vault server. This is a URL with a scheme, a hostname and a port but with no path. May be set via the `VAULT_ADDR` environment variable.
- `token` - (Required) Vault token that will be used by Terraform to authenticate. May be set via the `VAULT_TOKEN` environment variable. If none is otherwise supplied, Terraform will attempt to read it from `~/.vault-token` (where the vault command stores its current token). Terraform will issue itself a new token that is a child of the one given, with a short TTL to limit the exposure of any requested secrets. Note that the given token must have the update capability on the auth/token/create path in Vault in order to create child tokens.
- `ca_cert_file` - (Optional) Path to a file on local disk that will be used to validate the certificate presented by the Vault server. May be set via the `VAULT_CACERT` environment variable.
- `ca_cert_dir` - (Optional) Path to a directory on local disk that contains one or more certificate files that will be used to validate the certificate presented by the Vault server. May be set via the `VAULT_CAPATH` environment variable.
- `client_auth` - (Optional) A configuration block, described below, that provides credentials used by Terraform to authenticate with the Vault server. At present there is little reason to set this, because Terraform does not support the TLS certificate authentication mechanism.
- `skip_tls_verify` - (Optional) Set this to `true` to disable verification of the Vault server's TLS certificate. This is strongly discouraged except in prototype or development environments, since it exposes the possibility that Terraform can be tricked into writing secrets to a server controlled by an intruder. May be set via the `VAULT_SKIP_VERIFY` environment variable.
- `max_lease_ttl_seconds` - (Optional) Used as the duration for the intermediate Vault token Terraform issues itself, which in turn limits the duration of secret leases issued by Vault. Defaults to 20 minutes and may be set via the `TERRAFORM_VAULT_MAX_TTL` environment variable. See the section above on *Using Vault credentials in Terraform configuration* for the implications of this setting.

The `client_auth` configuration block accepts the following arguments:

- `cert_file` - (Required) Path to a file on local disk that contains the PEM-encoded certificate to present to the server.
- `key_file` - (Required) Path to a file on local disk that contains the PEM-encoded private key for which the authentication certificate was issued.

Example Usage

```
provider "vault" {
  # It is strongly recommended to configure this provider through the
  # environment variables described above, so that each user can have
  # separate credentials set in the environment.
  #
  # This will default to using $VAULT_ADDR
  # But can be set explicitly
  # address = "https://vault.example.net:8200"
}

resource "vault_generic_secret" "example" {
  path = "secret/foo"

  data_json = <<EOT
{
  "foo":   "bar",
  "pizza": "cheese"
}
EOT
}
```

vault_approle_auth_backend_role

Reads the Role ID of an AppRole from a Vault server.

Example Usage

```
data "vault_approle_auth_backend_role_id" "role" {
  backend   = "my-approle-backend"
  role_name = "my-role"
}

output "role-id" {
  value = "${data.vault_approle_auth_backend_role_id.role.role_id}"
}
```

Argument Reference

The following arguments are supported:

- `role_name` - (Required) The name of the role to retrieve the Role ID for.
- `backend` - (Optional) The unique name for the AppRole backend the role to retrieve a RoleID for resides in. Defaults to "approle".

Attributes Reference

In addition to the above arguments, the following attributes are exported:

- `role_id` - The RoleID of the role.

vault_aws_access_credentials

Reads AWS credentials from an AWS secret backend in Vault.

Important All data retrieved from Vault will be written in cleartext to state file generated by Terraform, will appear in the console output when Terraform runs, and may be included in plan files if secrets are interpolated into any resource attributes. Protect these artifacts accordingly. See the main provider documentation (/docs/providers/vault/index.html) for more details.

Example Usage

```
resource "vault_aws_secret_backend" "aws" {
  access_key = "AKIA....."
  secret_key = "SECRETKEYFROMAWS"
}

resource "vault_aws_secret_backend_role" "role" {
  backend = "${vault_aws_secret_backend.aws.path}"
  name    = "test"

  policy = <<EOT
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iam:*",
      "Resource": "*"
    }
  ]
}
EOT
}

# generally, these blocks would be in a different module
data "vault_aws_access_credentials" "creds" {
  backend = "${vault_aws_secret_backend.aws.path}"
  role    = "${vault_aws_secret_backend_role.role.name}"
}

provider "aws" {
  access_key = "${data.vault_aws_access_credentials.creds.access_key}"
  secret_key = "${data.vault_aws_access_credentials.creds.secret_key}"
}
```

Argument Reference

The following arguments are supported:

- `backend` - (Required) The path to the AWS secret backend to read credentials from, with no leading or trailing `/`s.
- `role` - (Required) The name of the AWS secret backend role to read credentials from, with no leading or trailing `/`s.

- `type` - (Optional) The type of credentials to read. Defaults to `"creds"`, which just returns an AWS Access Key ID and Secret Key. Can also be set to `"sts"`, which will return a security token in addition to the keys.

Attributes Reference

In addition to the arguments above, the following attributes are exported:

- `access_key` - The AWS Access Key ID returned by Vault.
- `secret_key` - The AWS Secret Key returned by Vault.
- `security_token` - The STS token returned by Vault, if any.
- `lease_id` - The lease identifier assigned by Vault.
- `lease_duration` - The duration of the secret lease, in seconds relative to the time the data was requested. Once this time has passed any plan generated with this data may fail to apply.
- `lease_start_time` - As a convenience, this records the current time on the computer where Terraform is running when the data is requested. This can be used to approximate the absolute time represented by `lease_duration`, though users must allow for any clock drift and response latency relative to the Vault server.
- `lease_renewable` - `true` if the lease can be renewed using Vault's `sys/renew/{lease-id}` endpoint. Terraform does not currently support lease renewal, and so it will request a new lease each time this data source is refreshed.

vault_generic_secret

Reads arbitrary data from a given path in Vault.

This resource is primarily intended to be used with Vault's "generic" secret backend (https://www.vaultproject.io/docs/secrets/generic/index.html), but it is also compatible with any other Vault endpoint that supports the `vault read` command.

Important All data retrieved from Vault will be written in cleartext to state file generated by Terraform, will appear in the console output when Terraform runs, and may be included in plan files if secrets are interpolated into any resource attributes. Protect these artifacts accordingly. See the main provider documentation (/docs/providers/vault/index.html) for more details.

Example Usage

```
data "vault_generic_secret" "rundeck_auth" {
  path = "secret/rundeck_auth"
}

# Rundeck Provider, for example
# For this example, in Vault there is a key named "auth_token" and the value is the token we need to keep
 secret.
# In general usage, replace "auth_token" with the key you wish to extract from Vault.

provider "rundeck" {
  url        = "http://rundeck.example.com/"
  auth_token = "${data.vault_generic_secret.rundeck_auth.data["auth_token"]}"
}
```

Argument Reference

The following arguments are supported:

- `path` - (Required) The full logical path from which to request data. To read data from the "generic" secret backend mounted in Vault by default, this should be prefixed with `secret/`. Reading from other backends with this data source is possible; consult each backend's documentation to see which endpoints support the `GET` method.

Required Vault Capabilities

Use of this resource requires the `read` capability on the given path.

Attributes Reference

The following attributes are exported:

- `data_json` - A string containing the full data payload retrieved from Vault, serialized in JSON format.

- `data` - A mapping whose keys are the top-level data keys returned from Vault and whose values are the corresponding values. This map can only represent string data, so any non-string values returned from Vault are serialized as JSON.
- `lease_id` - The lease identifier assigned by Vault, if any.
- `lease_duration` - The duration of the secret lease, in seconds relative to the time the data was requested. Once this time has passed any plan generated with this data may fail to apply.
- `lease_start_time` - As a convenience, this records the current time on the computer where Terraform is running when the data is requested. This can be used to approximate the absolute time represented by `lease_duration`, though users must allow for any clock drift and response latency relative to to the Vault server.
- `lease_renewable` - `true` if the lease can be renewed using Vault's `sys/renew/{lease-id}` endpoint. Terraform does not currently support lease renewal, and so it will request a new lease each time this data source is refreshed.

vault_kubernetes_auth_backend_config

Reads the Role of an Kubernetes from a Vault server. See the Vault documentation (https://www.vaultproject.io/api/auth/kubernetes/index.html#read-config) for more information.

Example Usage

```
data "vault_kubernetes_auth_backend_config" "config" {
  backend = "my-kubernetes-backend"
}

output "token_reviewer_jwt" {
  value = "${data.vault_kubernetes_auth_backend_config.config.token_reviewer_jwt}"
}
```

Argument Reference

The following arguments are supported:

- `backend` - (Optional) The unique name for the Kubernetes backend the config to retrieve Role attributes for resides in. Defaults to "kubernetes".

Attributes Reference

In addition to the above arguments, the following attributes are exported:

- `kubernetes_host` - Host must be a host string, a host:port pair, or a URL to the base of the Kubernetes API server.
- `kubernetes_ca_cert` - PEM encoded CA cert for use by the TLS client used to talk with the Kubernetes API.
- `pem_keys` - Optional list of PEM-formatted public keys or certificates used to verify the signatures of Kubernetes service account JWTs. If a certificate is given, its public key will be extracted. Not every installation of Kubernetes exposes these keys.

vault_kubernetes_auth_backend_role

Reads the Role of an Kubernetes from a Vault server. See the Vault documentation (https://www.vaultproject.io/api/auth/kubernetes/index.html#read-role) for more information.

Example Usage

```
data "vault_kubernetes_auth_backend_role" "role" {
  backend   = "my-kubernetes-backend"
  role_name = "my-role"
}

output "policies" {
  value = "${data.vault_kubernetes_auth_backend_role.role.policies}"
}
```

Argument Reference

The following arguments are supported:

- `role_name` - (Required) The name of the role to retrieve the Role attributes for.
- `backend` - (Optional) The unique name for the Kubernetes backend the role to retrieve Role attributes for resides in. Defaults to "kubernetes".

Attributes Reference

In addition to the above arguments, the following attributes are exported:

- `bound_service_account_names` - List of service account names able to access this role. If set to "" *all names are allowed, both this and bound_service_account_namespaces can not be ""*.
- `bound_service_account_namespaces` - List of namespaces allowed to access this role. If set to "" *all namespaces are allowed, both this and bound_service_account_names can not be set to ""*.
- `ttl` - The TTL period of tokens issued using this role in seconds.
- `max_ttl` - The maximum allowed lifetime of tokens issued in seconds using this role.
- `num_uses` - Number of times issued tokens can be used. Setting this to 0 or leaving it unset means unlimited uses.
- `period` - If set, indicates that the token generated using this role should never expire. The token should be renewed within the duration specified by this value. At each renewal, the token's TTL will be set to the value of this parameter.
- `policies` - Policies to be set on tokens issued using this role.

vault_approle_auth_backend_login

Logs into Vault using the AppRole auth backend. See the Vault documentation (https://www.vaultproject.io/docs/auth/approle.html) for more information.

Example Usage

```
resource "vault_auth_backend" "approle" {
  type = "approle"
}

resource "vault_approle_auth_backend_role" "example" {
  backend   = "${vault_auth_backend.approle.path}"
  role_name = "test-role"
  policies  = ["default", "dev", "prod"]
}

resource "vault_approle_auth_backend_role_secret_id" "id" {
  backend   = "${vault_auth_backend.approle.path}"
  role_name = "${vault_approle_auth_backend_role.example.role_name}"
}

resource "vault_approle_auth_backend_login" "login" {
  backend   = "${vault_auth_backend.approle.path}"
  role_id   = "${vault_approle_auth_backend_role.example.role_id}"
  secret_id = "${vault_approle_auth_backend_role_secret_id.id.secret_id}"
}
```

Argument Reference

The following arguments are supported:

- `role_id` - (Required) The ID of the role to log in with.
- `secret_id` - (Optional) The secret ID of the role to log in with. Required unless `bind_secret_id` is set to false on the role.
- `backend` - The unique path of the Vault backend to log in with.

Attributes Reference

In addition to the fields above, the following attributes are exported:

- `policies` - A list of policies applied to the token.
- `renewable` - Whether the token is renewable or not.
- `lease_duration` - How long the token is valid for, in seconds.
- `lease_started` - The date and time the lease started, in RFC 3339 format.
- `accessor` - The accessor for the token.

- `client_token` - The Vault token created.
- `metadata` - The metadata associated with the token.

vault_approle_auth_backend_role

Manages an AppRole auth backend role in a Vault server. See the Vault documentation (https://www.vaultproject.io/docs/auth/approle.html) for more information.

Example Usage

```
resource "vault_auth_backend" "approle" {
  type = "approle"
}

resource "vault_approle_auth_backend_role" "example" {
  backend   = "${vault_auth_backend.approle.path}"
  role_name = "test-role"
  policies  = ["default", "dev", "prod"]
}
```

Argument Reference

The following arguments are supported:

- `role_name` - (Required) The name of the role.
- `role_id` - (Optional) The RoleID of this role. If not specified, one will be auto-generated.
- `bind_secret_id` - (Optional) Whether or not to require `secret_id` to be presented when logging in using this AppRole. Defaults to `true`.
- `bound_cidr_list` - (Optional) If set, specifies blocks of IP addresses which can perform the login operation.
- `policies` - (Optional) An array of strings specifying the policies to be set on tokens issued using this role.
- `secret_id_num_uses` - (Optional) The number of times any particular SecretID can be used to fetch a token from this AppRole, after which the SecretID will expire. A value of zero will allow unlimited uses.
- `secret_id_ttl` - (Optional) The number of seconds after which any SecretID expires.
- `token_num_uses` - (Optional) The number of times issued tokens can be used. A value of 0 means unlimited uses.
- `token_ttl` - (Optional) The TTL period of tokens issued using this role, provided as a number of seconds.
- `token_max_ttl` - (Optional) The maximum allowed lifetime of tokens issued using this role, provided as a number of seconds.
- `period` - (Optional) If set, indicates that the token generated using this role should never expire. The token should be renewed within the duration specified by this value. At each renewal, the token's TTL will be set to the value of this field. The maximum allowed lifetime of token issued using this role. Specified as a number of seconds.
- `backend` - (Optional) The unique name of the auth backend to configure. Defaults to `approle`.

Attributes Reference

No additional attributes are exported by this resource.

Import

AppRole authentication backend roles can be imported using the `path`, e.g.

```
$ terraform import vault_approle_auth_backend_role.example auth/approle/role/test-role
```

vault_approle_auth_backend_role_secret_id

Manages an AppRole auth backend SecretID in a Vault server. See the Vault documentation (https://www.vaultproject.io/docs/auth/approle.html) for more information.

Example Usage

```
resource "vault_auth_backend" "approle" {
  type = "approle"
}

resource "vault_approle_auth_backend_role" "example" {
  backend   = "${vault_auth_backend.approle.path}"
  role_name = "test-role"
  policies  = ["default", "dev", "prod"]
}

resource "vault_approle_auth_backend_role_secret_id" "id" {
  backend   = "${vault_auth_backend.approle.path}"
  role_name = "${vault_approle_auth_backend_role.example.role_name}"

  metadata = <<EOT
{
  "hello": "world"
}
EOT
}
```

Argument Reference

The following arguments are supported:

- `role_name` - (Required) The name of the role to create the SecretID for.
- `metadata` - (Optional) A JSON-encoded string containing metadata in key-value pairs to be set on tokens issued with this SecretID.
- `cidr_list` - (Optional) If set, specifies blocks of IP addresses which can perform the login operation using this SecretID.
- `secret_id` - (Optional) The SecretID to be created. If set, uses "Push" mode. Defaults to Vault auto-generating SecretIDs.

Attributes Reference

In addition to the fields above, the following attributes are exported:

- `accessor` - The unique ID for this SecretID that can be safely logged.

vault_audit

Example Usage (file audit device)

```
resource "vault_audit" "test" {
  type = "file"

  options = {
    file_path = "C:/temp/audit.txt"
  }
}
```

Example Usage (socket audit device)

```
resource "vault_audit" "test" {
  type = "socket"
  path = "app_socket"

  options = {
    address     = "127.0.0.1:8000"
    socket_type = "tcp"
    description = "application x socket"
  }
}
```

Argument Reference

The following arguments are supported:

- `type` - (Required) Type of the audit device, such as 'file'.
- `path` - (optional) The path to mount the audit device. This defaults to the type.
- `description` - (Optional) Human-friendly description of the audit device.
- `options` - (Required) Configuration options to pass to the audit device itself.

For a reference of the device types and their options, consult the Vault documentation.
(https://www.vaultproject.io/docs/audit/index.html)

Attributes Reference

No additional attributes are exported by this resource.

Import

Audit devices can be imported using the `path`, e.g.

```
$ terraform import vault_audit.test syslog
```

vault_auth_backend

Example Usage

```
resource "vault_auth_backend" "example" {
  type = "github"
}
```

Argument Reference

The following arguments are supported:

- `type` - (Required) The name of the auth method type
- `path` - (Optional) The path to mount the auth method — this defaults to the name of the type
- `description` - (Optional) A description of the auth method
- `default_lease_ttl_seconds` - (Optional) The default lease duration in seconds.
- `max_lease_ttl_seconds` - (Optional) The maximum lease duration in seconds.
- `listing_visibility` - (Optional) Speficies whether to show this mount in the UI-specific listing endpoint.
- `local` - (Optional) Specifies if the auth method is local only.

Attributes Reference

In addition to the fields above, the following attributes are exported:

- `accessor` - The accessor for this auth method

Import

Auth methods can be imported using the `path`, e.g.

```
$ terraform import vault_auth_backend.example github
```

vault_aws_auth_backend_cert

Manages a certificate to be used with an AWS Auth Backend in Vault.

This resource sets the AWS public key and the type of document that can be verified against the key that Vault can then use to verify the instance identity documents making auth requests.

For more information, see the Vault docs (https://www.vaultproject.io/api/auth/aws/index.html#configure-client).

Important All data provided in the resource configuration will be written in cleartext to state and plan files generated by Terraform, and will appear in the console output when Terraform runs. Protect these artifacts accordingly. See the main provider documentation (/docs/providers/vault/index.html) for more details.

Example Usage

```
resource "vault_auth_backend" "aws" {
  type = "aws"
}

resource "vault_aws_auth_backend_cert" "cert" {
  backend         = "${vault_auth_backend.aws.path}"
  cert_name       = "my-cert"
  aws_public_cert = "${file("${path.module}/aws_public_key.crt})"
  type            = "pkcs7"
}
```

Argument Reference

The following arguments are supported:

- `cert_name` - (Required) The name of the certificate.
- `aws_public_cert` - (Required) The Base64 encoded AWS Public key required to verify PKCS7 signature of the EC2 instance metadata. You can find this key in the AWS documentation (http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-identity-documents.html).
- `type` - (Optional) Either "pkcs7" or "identity", indicating the type of document which can be verified using the given certificate. Defaults to "pkcs7".
- `backend` - (Optional) The path the AWS auth backend being configured was mounted at. Defaults to `aws`.

Attributes Reference

No additional attributes are exported by this resource.

Import

AWS auth backend certificates can be imported using `auth/`, the `backend` path, `/config/certificate/`, and the `cert_name` e.g.

```
$ terraform import vault_aws_auth_backend_cert.example auth/aws/config/certificate/my-cert
```

vault_aws_auth_backend_client

Configures the client used by an AWS Auth Backend in Vault.

This resource sets the access key and secret key that Vault will use when making API requests on behalf of an AWS Auth Backend. It can also be used to override the URLs Vault uses when making those API requests.

For more information, see the Vault docs (https://www.vaultproject.io/api/auth/aws/index.html#configure-client).

Important All data provided in the resource configuration will be written in cleartext to state and plan files generated by Terraform, and will appear in the console output when Terraform runs. Protect these artifacts accordingly. See the main provider documentation (/docs/providers/vault/index.html) for more details.

Example Usage

```
resource "vault_auth_backend" "example" {
  type = "aws"
}

resource "vault_aws_auth_backend_client" "example" {
  backend    = "${vault_auth_backend.example.path}"
  access_key = "INSERT_AWS_ACCESS_KEY"
  secret_key = "INSERT_AWS_SECRET_KEY"
}
```

Argument Reference

The following arguments are supported:

- `backend` - (Optional) The path the AWS auth backend being configured was mounted at. Defaults to `aws`.
- `access_key` - (Optional) The AWS access key that Vault should use for the auth backend.
- `secret_key` - (Optional) The AWS secret key that Vault should use for the auth backend.
- `ec2_endpoint` - (Optional) Override the URL Vault uses when making EC2 API calls.
- `iam_endpoint` - (Optional) Override the URL Vault uses when making IAM API calls.
- `sts_endpoint` - (Optional) Override the URL Vault uses when making STS API calls.
- `iam_server_id_header_value` - (Optional) The value to require in the `X-Vault-AWS-IAM-Server-ID` header as part of `GetCallerIdentity` requests that are used in the IAM auth method.

Attributes Reference

No additional attributes are exported by this resource.

Import

AWS auth backend clients can be imported using `auth/`, the `backend` path, and `/config/client` e.g.

```
$ terraform import vault_aws_auth_backend_client.example auth/aws/config/client
```

vault_aws_auth_backend_identity_whitelist

Configures the periodic tidying operation of the whitelisted identity entries.

For more information, see the Vault docs (https://www.vaultproject.io/api/auth/aws/index.html#configure-identity-whitelist-tidy-operation).

Example Usage

```
resource "vault_auth_backend" "example" {
  type = "aws"
}

resource "vault_aws_auth_backend_identity_whitelist" "example" {
  backend       = "${vault_auth_backend.example.path}"
  safety_buffer = 3600
}
```

Argument Reference

The following arguments are supported:

- `backend` - (Optional) The path of the AWS backend being configured.
- `safety_buffer` - (Optional) The amount of extra time, in minutes, that must have passed beyond the roletag expiration, before it is removed from the backend storage.
- `disable_periodic_tidy` - (Optional) If set to true, disables the periodic tidying of the identity-whitelist entries.

Attributes Reference

No additional attributes are exported by this resource.

Import

AWS auth backend identity whitelists can be imported using `auth/`, the `backend` path, and `/config/tidy/identity-whitelist` e.g.

```
$ terraform import vault_aws_auth_backend_identity_whitelist.example auth/aws/config/tidy/identity-whitel
ist
```

vault_aws_auth_backend_login

Logs into a Vault server using an AWS auth backend. Login can be accomplished using a signed identity request from IAM or using ec2 instance metadata. For more information, see the Vault documentation (https://www.vaultproject.io/docs/auth/aws.html).

Example Usage

```
resource "vault_auth_backend" "aws" {
  type = "aws"
}

resource "vault_aws_auth_backend_client" "example" {
  backend    = "${vault_auth_backend.aws.path}"
  access_key = "123456789012"
  secret_key = "AWSSECRETKEYGOESHERE"
}

resource "vault_aws_auth_backend_role" "example" {
  backend                         = "${vault_auth_backend.aws.path}"
  role                            = "test-role"
  auth_type                       = "ec2"
  bound_ami_id                    = "ami-8c1be5f6"
  bound_account_id                = "123456789012"
  bound_vpc_id                    = "vpc-b61106d4"
  bound_subnet_id                 = "vpc-133128f1"
  bound_iam_instance_profile_arn = "arn:aws:iam::123456789012:instance-profile/MyProfile"
  ttl                             = 60
  max_ttl                         = 120
  policies                        = ["default", "dev", "prod"]

  depends_on                      = ["vault_aws_auth_backend_client.example"]
}

resource "vault_aws_auth_backend_login" "example" {
  backend   = "${vault_auth_backend.example.path}"
  role      = "${vault_aws_auth_backend_role.example.role}"
  identity  = "BASE64ENCODEDIDENTITYDOCUMENT"
  signature = "BASE64ENCODEDSHA256IDENTITYDOCUMENTSIGNATURE"
}
```

Argument Reference

The following arguments are supported:

- `backend` - (Optional) The unique name of the AWS auth backend. Defaults to 'aws'.
- `role` - (Optional) The name of the AWS auth backend role to create tokens against.
- `identity` - (Optional) The base64-encoded EC2 instance identity document to authenticate with. Can be retrieved from the EC2 metadata server.
- `signature` - (Optional) The base64-encoded SHA256 RSA signature of the instance identity document to authenticate with, with all newline characters removed. Can be retrieved from the EC2 metadata server.

- `pkcs7` - (Optional) The PKCS#7 signature of the identity document to authenticate with, with all newline characters removed. Can be retrieved from the EC2 metadata server.
- `nonce` - (Optional) The unique nonce to be used for login requests. Can be set to a user-specified value, or will contain the server-generated value once a token is issued. EC2 instances can only acquire a single token until the whitelist is tidied again unless they keep track of this nonce.
- `iam_http_request_method` - (Optional) The HTTP method used in the signed IAM request.
- `iam_request_url` - (Optional) The base64-encoded HTTP URL used in the signed request.
- `iam_request_body` - (Optional) The base64-encoded body of the signed request.
- `iam_request_headers` - (Optional) The base64-encoded, JSON serialized representation of the GetCallerIdentity HTTP request headers.

Attributes Reference

In addition to the fields above, the following attributes are also exposed:

- `lease_duration` - The duration in seconds the token will be valid, relative to the time in `lease_start_time`.
- `lease_start_time` - The approximate time at which the token was created, using the clock of the system where Terraform was running.
- `renewable` - Set to true if the token can be extended through renewal.
- `metadata` - A map of information returned by the Vault server about the authentication used to generate this token.
- `auth_type` - The authentication type used to generate this token.
- `policies` - The Vault policies assigned to this token.
- `accessor` - The token's accessor.
- `client_token` - The token returned by Vault.

vault_aws_auth_backend_role

Manages an AWS auth backend role in a Vault server. Roles constrain the instances or principals that can perform the login operation against the backend. See the Vault documentation (https://www.vaultproject.io/docs/auth/aws.html) for more information.

Example Usage

```
resource "vault_auth_backend" "aws" {
  type = "aws"
}

resource "vault_aws_auth_backend_role" "example" {
  backend                          = "${vault_auth_backend.aws.path}"
  role                             = "test-role"
  auth_type                        = "iam"
  bound_ami_ids                    = ["ami-8c1be5f6"]
  bound_account_ids                = ["123456789012"]
  bound_vpc_ids                    = ["vpc-b61106d4"]
  bound_subnet_ids                 = ["vpc-133128f1"]
  bound_iam_role_arns              = ["arn:aws:iam::123456789012:role/MyRole"]
  bound_iam_instance_profile_arns = ["arn:aws:iam::123456789012:instance-profile/MyProfile"]
  inferred_entity_type             = "ec2_instance"
  inferred_aws_region              = "us-east-1"
  ttl                              = 60
  max_ttl                          = 120
  policies                         = ["default", "dev", "prod"]
}
```

Argument Reference

The following arguments are supported:

- `role` - (Required) The name of the role.
- `auth_type` - (Optional) The auth type permitted for this role. Valid choices are `ec2` and `iam`. Defaults to `iam`.
- `bound_ami_ids` - (Optional) If set, defines a constraint on the EC2 instances that can perform the login operation that they should be using the AMI ID specified by this field. `auth_type` must be set to `ec2` or `inferred_entity_type` must be set to `ec2_instance` to use this constraint.
- `bound_account_ids` - (Optional) If set, defines a constraint on the EC2 instances that can perform the login operation that they should be using the account ID specified by this field. `auth_type` must be set to `ec2` or `inferred_entity_type` must be set to `ec2_instance` to use this constraint.
- `bound_regions` - (Optional) If set, defines a constraint on the EC2 instances that can perform the login operation that the region in their identity document must match the one specified by this field. `auth_type` must be set to `ec2` or `inferred_entity_type` must be set to `ec2_instance` to use this constraint.
- `bound_vpc_ids` - (Optional) If set, defines a constraint on the EC2 instances that can perform the login operation that they be associated with the VPC ID that matches the value specified by this field. `auth_type` must be set to `ec2` or `inferred_entity_type` must be set to `ec2_instance` to use this constraint.

- `bound_subnet_ids` - (Optional) If set, defines a constraint on the EC2 instances that can perform the login operation that they be associated with the subnet ID that matches the value specified by this field. `auth_type` must be set to `ec2` or `inferred_entity_type` must be set to `ec2_instance` to use this constraint.
- `bound_iam_role_arns` - (Optional) If set, defines a constraint on the EC2 instances that can perform the login operation that they must match the IAM role ARN specified by this field. `auth_type` must be set to `ec2` or `inferred_entity_type` must be set to `ec2_instance` to use this constraint.
- `bound_iam_instance_profile_arns` - (Optional) If set, defines a constraint on the EC2 instances that can perform the login operation that they must be associated with an IAM instance profile ARN which has a prefix that matches the value specified by this field. The value is prefix-matched as though it were a glob ending in `*`. `auth_type` must be set to `ec2` or `inferred_entity_type` must be set to `ec2_instance` to use this constraint.
- `role_tag` - (Optional) If set, enable role tags for this role. The value set for this field should be the key of the tag on the EC2 instance. `auth_type` must be set to `ec2` or `inferred_entity_type` must be set to `ec2_instance` to use this constraint.
- `bound_iam_principal_arns` - (Optional) If set, defines the IAM principal that must be authenticated when `auth_type` is set to `iam`. Wildcards are supported at the end of the ARN.
- `inferred_entity_type` - (Optional) If set, instructs Vault to turn on inferencing. The only valid value is `ec2_instance`, which instructs Vault to infer that the role comes from an EC2 instance in an IAM instance profile. This only applies when `auth_type` is set to `iam`.
- `inferred_aws_region` - (Optional) When `inferred_entity_type` is set, this is the region to search for the inferred entities. Required if `inferred_entity_type` is set. This only applies when `auth_type` is set to `iam`.
- `resolve_aws_unique_ids` - (Optional) If set to `true`, the `bound_iam_principal_arns` are resolved to AWS Unique IDs (http://docs.aws.amazon.com/IAM/latest/UserGuide/reference_identifiers.html#identifiers-unique-ids) for the bound principal ARN. This field is ignored when a `bound_iam_principal_arn` ends in a wildcard. Resolving to unique IDs more closely mimics the behavior of AWS services in that if an IAM user or role is deleted and a new one is recreated with the same name, those new users or roles won't get access to roles in Vault that were permissioned to the prior principals of the same name. Defaults to `true`. Once set to `true`, this cannot be changed to `false`--the role must be deleted and recreated, with the value set to `true`.
- `ttl` - (Optional) The TTL period of tokens issued using this role, provided as a number of seconds.
- `max_ttl` - (Optional) The maximum allowed lifetime of tokens issued using this role, provided as a number of seconds.
- `period` - (Optional) If set, indicates that the token generated using this role should never expire. The token should be renewed within the duration specified by this value. At each renewal, the token's TTL will be set to the value of this field. The maximum allowed lifetime of token issued using this role. Specified as a number of seconds.
- `policies` - (Optional) An array of strings specifying the policies to be set on tokens issued using this role.
- `allow_instance_migration` - (Optional) If set to `true`, allows migration of the underlying instance where the client resides.
- `disallow_reauthentication` - (Optional) IF set to `true`, only allows a single token to be granted per instance ID. This can only be set when `auth_type` is set to `ec2`.

Attributes Reference

No additional attributes are exported by this resource.

Import

AWS auth backend roles can be imported using `auth/`, the `backend` path, `/role/`, and the `role` name e.g.

```
$ terraform import vault_aws_auth_backend_role.example auth/aws/role/test-role
```

vault_aws_auth_backend_role_tag

Reads role tag information from an AWS auth backend in Vault.

Example Usage

```
resource "vault_auth_backend" "aws" {
  path = "%s"
  type = "aws"
}

resource "vault_aws_auth_backend_role" "role" {
  backend          = "${vault_auth_backend.aws.path}"
  role             = "%s"
  auth_type        = "ec2"
  bound_account_id = "123456789012"
  policies         = ["dev", "prod", "qa", "test"]
  role_tag         = "VaultRoleTag"
}

resource "vault_aws_auth_backend_role_tag" "test" {
  backend     = "${vault_auth_backend.aws.path}"
  role        = "${vault_aws_auth_backend_role.role.role}"
  policies    = ["prod", "dev", "test"]
  max_ttl     = "1h"
  instance_id = "i-1234567"
}
```

Argument Reference

The following arguments are supported:

- `role` - (Required) The name of the AWS auth backend role to read role tags from, with no leading or trailing `/`s.
- `backend` - (Optional) The path to the AWS auth backend to read role tags from, with no leading or trailing `/`s. Defaults to "aws".
- `policies` - (Optional) The policies to be associated with the tag. Must be a subset of the policies associated with the role.
- `max_ttl` - (Optional) The maximum TTL of the tokens issued using this role.
- `instance_id` - (Optional) Instance ID for which this tag is intended for. If set, the created tag can only be used by the instance with the given ID.
- `allow_instance_migration` - (Optional) If set, allows migration of the underlying instances where the client resides. Use with caution.
- `disallow_reauthentication` - (Optional) If set, only allows a single token to be granted per instance ID.

Attributes Reference

In addition to the arguments above, the following attributes are exported:

- `tag_key` - The key of the role tag.
- `tag_value` - The value to set the role key.

layout: "vault" page_title: "Vault: vault_aws_auth_backend_roletag_blacklist resource" sidebar_current: "docs-vault-resource-aws-auth-backend-roletag-blacklist description: |-

Configures the periodic tidying operation of the blacklisted role tag entries.

vault_aws_auth_backend_roletag_blacklist

Configures the periodic tidying operation of the blacklisted role tag entries.

Example Usage

```
resource "vault_auth_backend" "example" {
  type = "aws"
}

resource "vault_aws_auth_backend_roletag_blacklist" "example" {
  backend       = "${vault_auth_backend.example.path}"
  safety_buffer = 360
}
```

Argument Reference

The following arguments are supported:

- `backend` - (Required) The path the AWS auth backend being configured was mounted at.
- `safety_buffer` - (Oprtional) The amount of extra time that must have passed beyond the roletag expiration, before it is removed from the backend storage. Defaults to 259,200 seconds, or 72 hours.
- `disable_periodic_tidy` - (Optional) If set to true, disables the periodic tidying of the roletag blacklist entries. Defaults to false.

Attributes Reference

No additional attributes are exported by this resource.

vault_aws_auth_backend_sts_role

Manages an STS role in a Vault server. STS roles are mappings between account IDs and STS ARNs. When a login attempt is made from an EC2 instance in the account ID specified, the associated STS role will be used to verify the request. For more information, see the Vault documentation (https://www.vaultproject.io/docs/auth/aws.html#cross-account-access).

Important All data provided in the resource configuration will be written in cleartext to state and plan files generated by Terraform, and will appear in the console output when Terraform runs. Protect these artifacts accordingly. See the main provider documentation (/docs/providers/index.html) for more details.

Example Usage

```
resource "vault_auth_backend" "aws" {
  type = "aws"
}

resource "vault_aws_auth_backend_sts_role" "role" {
  backend    = "${vault_auth_backend.aws.path}"
  account_id = "1234567890"
  sts_role   = "arn:aws:iam::1234567890:role/my-role"
}
```

Argument Reference

The following arguments are supported:

- `account_id` - (Optional) The AWS account ID to configure the STS role for.
- `sts_role` - (Optional) The STS role to assume when verifying requests made by EC2 instances in the account specified by `account_id`.
- `backend` - (Optional) The path the AWS auth backend being configured was mounted at. Defaults to `aws`.

Attributes Reference

No additional attributes are exported by this resource.

Import

AWS auth backend STS roles can be imported using `auth/`, the `backend` path, `/config/sts/`, and the `account_id` e.g.

```
$ terraform import vault_aws_auth_backend_sts_role.example auth/aws/config/sts/1234567890
```

vault_aws_secret_backend

Creates an AWS Secret Backend for Vault. AWS secret backends can then issue AWS access keys and secret keys, once a role has been added to the backend.

Important All data provided in the resource configuration will be written in cleartext to state and plan files generated by Terraform, and will appear in the console output when Terraform runs. Protect these artifacts accordingly. See the main provider documentation (/docs/providers/vault/index.html) for more details.

Example Usage

```
resource "vault_aws_secret_backend" "aws" {
  access_key = "AKIA....."
  secret_key = "AWS secret key"
}
```

Argument Reference

The following arguments are supported:

- `access_key` - (Required) The AWS Access Key ID this backend should use to issue new credentials.
- `secret_key` - (Required) The AWS Secret Key this backend should use to issue new credentials.

Important Because Vault does not support reading the configured credentials back from the API, Terraform cannot detect and correct drift on `access_key` or `secret_key`. Changing the values, however, *will* overwrite the previously stored values.

- `region` - (Optional) The AWS region for API calls. Defaults to `us-east-1`.
- `path` - (Optional) The unique path this backend should be mounted at. Must not begin or end with a `/`. Defaults to `aws`.
- `description` - (Optional) A human-friendly description for this backend.
- `default_lease_ttl_seconds` - (Optional) The default TTL for credentials issued by this backend.
- `max_lease_ttl_seconds` - (Optional) The maximum TTL that can be requested for credentials issued by this backend.

Attributes Reference

No additional attributes are exported by this resource.

Import

AWS secret backends can be imported using the `path`, e.g.

```
$ terraform import vault_aws_secret_backend.aws aws
```

vault_aws_secret_backend_role

Creates a role on an AWS Secret Backend for Vault. Roles are used to map credentials to the policies that generated them.

Important All data provided in the resource configuration will be written in cleartext to state and plan files generated by Terraform, and will appear in the console output when Terraform runs. Protect these artifacts accordingly. See the main provider documentation (/docs/providers/vault/index.html) for more details.

Example Usage

```
resource "vault_aws_secret_backend" "aws" {
  access_key = "AKIA....."
  secret_key = "AWS secret key"
}

resource "vault_aws_secret_backend_role" "role" {
  backend = "${vault_aws_secret_backend.aws.path}"
  name    = "deploy"

  policy = <<EOT
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iam:*",
      "Resource": "*"
    }
  ]
}
EOT
}
```

Argument Reference

The following arguments are supported:

- `backend` - (Required) The path the AWS secret backend is mounted at, with no leading or trailing /s.
- `name` - (Required) The name to identify this role within the backend. Must be unique within the backend.
- `policy` - (Optional) The JSON-formatted policy to associate with this role. Either `policy` or `policy_arn` must be specified.
- `policy_arn` - (Optional) The ARN for a pre-existing policy to associate with this role. Either `policy` or `policy_arn` must be specified.

Attributes Reference

No additional attributes are exported by this resource.

Import

RabbitMQ secret backend roles can be imported using the `path`, e.g.

```
$ terraform import vault_aws_secret_backend_role.role aws/roles/deploy
```

vault_cert_auth_backend_role

Provides a resource to create a role in an Cert auth backend within Vault (https://www.vaultproject.io/docs/auth/cert.html).

Example Usage

```
resource "vault_auth_backend" "cert" {
    path = "cert"
    type = "cert"
}

resource "vault_cert_auth_backend_role" "cert" {
    backend       = "${vault_auth_backend.cert.path}"
    allowed_names = ["foo.example.org", "baz.example.org"]
    ttl           = 300
    max_ttl       = 600
    policies      = ["foo"]
}
```

Argument Reference

The following arguments are supported:

- `name` - (Required) Name of the role
- `certificate` - (Required) CA certificate used to validate client certificates
- `allowed_names` - (Optional) Allowed subject names for authenticated client certificates
- `required_exwtensions` - (Optional) TLS extensions required on client certificates
- `ttl` - (Optional) Default TTL of tokens issued by the backend
- `max_ttl` - (Optional) Maximum TTL of tokens issued by the backend
- `period` - (Optional) Duration in seconds for token. If set, the issued token is a periodic token.
- `policies` - (Optional) Policies to grant on the issued token
- `display_name` - (Optional) The name to display on tokens issued under this role.
- `backend` - (Optional) Path to the mounted Cert auth backend

For more details on the usage of each argument consult the Vault Cert API documentation (https://www.vaultproject.io/api/auth/cert/index.html).

Attribute Reference

No additional attributes are exposed by this resource.

vault_consul_secret_backend

Creates a Consul Secret Backend for Vault. Consul secret backends can then issue Consul tokens, once a role has been added to the backend.

Important All data provided in the resource configuration will be written in cleartext to state and plan files generated by Terraform, and will appear in the console output when Terraform runs. Protect these artifacts accordingly. See the main provider documentation (/docs/providers/vault/index.html) for more details.

Example Usage

```
resource "vault_consul_secret_backend" "test" {
  path        = "consul"
  description = "Manages the Consul backend"

  address = "127.0.0.1:8500"
  token   = "4240861b-ce3d-8530-115a-521ff070dd29"
}
```

Argument Reference

The following arguments are supported:

- `token` - (Required) The Consul management token this backend should use to issue new tokens.

Important Because Vault does not support reading the configured token back from the API, Terraform cannot detect and correct drift on `token`. Changing the value, however, *will* overwrite the previously stored values.

- `backend` - (Optional) The unique location this backend should be mounted at. Must not begin or end with a `/`. Defaults to `consul`.
- `description` - (Optional) A human-friendly description for this backend.
- `address` - (Required) Specifies the address of the Consul instance, provided as "host:port" like "127.0.0.1:8500".
- `scheme` - (Optional) Specifies the URL scheme to use. Defaults to `http`.
- `default_lease_ttl_seconds` - (Optional) The default TTL for credentials issued by this backend.
- `max_lease_ttl_seconds` - (Optional) The maximum TTL that can be requested for credentials issued by this backend.

Attributes Reference

No additional attributes are exported by this resource.

Import

Consul secret backends can be imported using the `path`, e.g.

```
$ terraform import vault_consul_secret_backend.example consul
```

vault_database_secret_backend_connection

Creates a Database Secret Backend connection in Vault. Database secret backend connections can be used to generate dynamic credentials for the database.

Important All data provided in the resource configuration will be written in cleartext to state and plan files generated by Terraform, and will appear in the console output when Terraform runs. Protect these artifacts accordingly. See the main provider documentation (/docs/providers/vault/index.html) for more details.

Example Usage

```
resource "vault_mount" "db" {
  path = "postgres"
  type = "database"
}

resource "vault_database_secret_backend_connection" "postgres" {
  backend       = "${vault_mount.db.path}"
  name          = "postgres"
  allowed_roles = ["dev", "prod"]

  postgresql {
    connection_url = "postgres://username:password@host:port/database"
  }
}
```

Argument Reference

The following arguments are supported:

- `name` - (Required) A unique name to give the database connection.
- `backend` - (Required) The unique name of the Vault mount to configure.
- `verify_connection` - (Optional) Whether the connection should be verified on initial configuration or not.
- `allowed_roles` - (Optional) A list of roles that are allowed to use this connection.
- `cassandra` - (Optional) A nested block containing configuration options for Cassandra connections.
- `mongodb` - (Optional) A nested block containing configuration options for MongoDB connections.
- `hana` - (Optional) A nested block containing configuration options for SAP HanaDB connections.
- `mssql` - (Optional) A nested block containing configuration options for MSSQL connections.
- `mysql` - (Optional) A nested block containing configuration options for MySQL connections.
- `mysql_rds` - (Optional) A nested block containing configuration options for RDS MySQL connections.
- `mysql_aurora` - (Optional) A nested block containing configuration options for Aurora MySQL connections.

- `mysql_legacy` - (Optional) A nested block containing configuration options for legacy MySQL connections.
- `postgresql` - (Optional) A nested block containing configuration options for PostgreSQL connections.
- `oracle` - (Optional) A nested block containing configuration options for Oracle connections.

Exactly one of the nested blocks of configuration options must be supplied.

Cassandra Configuration Options

- `hosts` - (Required) The hosts to connect to.
- `username` - (Required) The username to authenticate with.
- `password` - (Required) The password to authenticate with.
- `port` - (Optional) The default port to connect to if no port is specified as part of the host.
- `tls` - (Optional) Whether to use TLS when connecting to Cassandra.
- `insecure_tls` - (Optional) Whether to skip verification of the server certificate when using TLS.
- `pem_bundle` - (Optional) Concatenated PEM blocks configuring the certificate chain.
- `pem_json` - (Optional) A JSON structure configuring the certificate chain.
- `protocol_version` - (Optional) The CQL protocol version to use.
- `connect_timeout` - (Optional) The number of seconds to use as a connection timeout.

MongoDB Configuration Options

- `connection_url` - (Required) A URL containing connection information. See the Vault docs (https://www.vaultproject.io/api/secret/databases/mongodb.html#sample-payload) for an example.

SAP HanaDB Configuration Options

- `connection_url` - (Required) A URL containing connection information. See the Vault docs (https://www.vaultproject.io/api/secret/databases/hanadb.html#sample-payload) for an example.
- `max_open_connections` - (Optional) The maximum number of open connections to use.
- `max_idle_connections` - (Optional) The maximum number of idle connections to maintain.
- `max_connection_lifetime` - (Optional) The maximum number of seconds to keep a connection alive for.

MSSQL Configuration Options

- `connection_url` - (Required) A URL containing connection information. See the Vault docs (https://www.vaultproject.io/api/secret/databases/mssql.html#sample-payload) for an example.
- `max_open_connections` - (Optional) The maximum number of open connections to use.

- `max_idle_connections` - (Optional) The maximum number of idle connections to maintain.
- `max_connection_lifetime` - (Optional) The maximum number of seconds to keep a connection alive for.

MySQL Configuration Options

- `connection_url` - (Required) A URL containing connection information. See the Vault docs (https://www.vaultproject.io/api/secret/databases/mysql-maria.html#sample-payload) for an example.
- `max_open_connections` - (Optional) The maximum number of open connections to use.
- `max_idle_connections` - (Optional) The maximum number of idle connections to maintain.
- `max_connection_lifetime` - (Optional) The maximum number of seconds to keep a connection alive for.

PostgreSQL Configuration Options

- `connection_url` - (Required) A URL containing connection information. See the Vault docs (https://www.vaultproject.io/api/secret/databases/postgresql.html#sample-payload) for an example.
- `max_open_connections` - (Optional) The maximum number of open connections to use.
- `max_idle_connections` - (Optional) The maximum number of idle connections to maintain.
- `max_connection_lifetime` - (Optional) The maximum number of seconds to keep a connection alive for.

Oracle Configuration Options

- `connection_url` - (Required) A URL containing connection information. See the Vault docs (https://www.vaultproject.io/api/secret/databases/oracle.html#sample-payload) for an example.
- `max_open_connections` - (Optional) The maximum number of open connections to use.
- `max_idle_connections` - (Optional) The maximum number of idle connections to maintain.
- `max_connection_lifetime` - (Optional) The maximum number of seconds to keep a connection alive for.

Attributes Reference

No additional attributes are exported by this resource.

Import

Database secret backend connections can be imported using the `backend`, `/config/`, and the `name` e.g.

```
$ terraform import vault_database_secret_backend_connection.example postgres/config/postgres
```

vault_database_secret_backend_role

Creates a Database Secret Backend role in Vault. Database secret backend roles can be used to generate dynamic credentials for the database.

Important All data provided in the resource configuration will be written in cleartext to state and plan files generated by Terraform, and will appear in the console output when Terraform runs. Protect these artifacts accordingly. See the main provider documentation (/docs/providers/vault/index.html) for more details.

Example Usage

```
resource "vault_mount" "db" {
  path = "postgres"
  type = "database"
}

resource "vault_database_secret_backend_connection" "postgres" {
  backend       = "${vault_mount.db.path}"
  name          = "postgres"
  allowed_roles = ["dev", "prod"]

  postgresql {
    connection_url = "postgres://username:password@host:port/database"
  }
}

resource "vault_database_secret_backend_role" "role" {
  backend             = "${vault_mount.db.path}"
  name                = "my-role"
  db_name             = "${vault_database_secret_backend_connection.postgres.name}"
  creation_statements = "CREATE ROLE \"{{name}}\" WITH LOGIN PASSWORD '{{password}}' VALID UNTIL '{{expir
ation}}';"
}
```

Argument Reference

The following arguments are supported:

- `name` - (Required) A unique name to give the role.
- `backend` - (Required) The unique name of the Vault mount to configure.
- `db_name` - (Required) The unique name of the database connection to use for the role.
- `creation_statements` - (Required) The database statements to execute when creating a user.
- `revocation_statements` - (Optional) The database statements to execute when revoking a user.
- `rollback_statements` - (Optional) The database statements to execute when rolling back creation due to an error.
- `renew_statements` - (Optional) The database statements to execute when renewing a user.

- `default_ttl` - (Optional) The default number of seconds for leases for this role.
- `max_ttl` - (Optional) The maximum number of seconds for leases for this role.

Attributes Reference

No additional attributes are exported by this resource.

Import

Database secret backend roles can be imported using the `backend`, `/roles/`, and the `name` e.g.

```
$ terraform import vault_database_secret_backend_role.example postgres/roles/my-role
```

vault_gcp_auth_backend\

Provides a resource to configure the GCP auth backend within Vault (https://www.vaultproject.io/docs/auth/gcp.html).

Example Usage

```
resource "vault_gcp_auth_backend" "gcp" {
    credentials  = "${file("vault-gcp-credentials.json")}"
}
```

Argument Reference

The following arguments are supported:

- `credentials` - (Required) A JSON string containing the contents of a GCP credentials file.

For more details on the usage of each argument consult the Vault GCP API documentation (https://www.vaultproject.io/api/auth/gcp/index.html#configure).

Attribute Reference

In addition to the fields above, the following attributes are also exposed:

- `client_id` - The Client ID of the credentials
- `private_key_id` - The ID of the private key from the credentials
- `project_id` - The GCP Project ID
- `client_email` - The clients email assosiated with the credentials

vault_gcp_auth_backend_role

Provides a resource to create a role in an GCP auth backend within Vault (https://www.vaultproject.io/docs/auth/gcp.html).

Example Usage

```
resource "vault_auth_backend" "gcp" {
    path = "gcp"
    type = "gcp"
}

resource "vault_gcp_auth_backend_role" "gcp" {
    backend                = "${vault_auth_backend.cert.path}"
    project_id             = "foo-bar-baz"
    bound_service_accounts = ["database-server@foo-bar-baz.iam.gserviceaccount.com"]
    policies               = ["database-server"]

}
```

Argument Reference

The following arguments are supported:

- `role` - (Required) Name of the GCP role
- `type` - (Required) Type of GCP authentication role (either `gce` or `iam`)
- `project_id` - (Required) GCP Project that the role exists within
- `ttl` - (Optional) Default TTL of tokens issued by the backend
- `max_ttl` - (Optional) Maximum TTL of tokens issued by the backend
- `period` - (Optional) Duration in seconds for token. If set, the issued token is a periodic token.
- `policies` - (Optional) Policies to grant on the issued token
- `backend` - (Optional) Path to the mounted GCP auth backend
- `bound_service_accounts` - (Optional) GCP Service Accounts allowed to issue tokens under this role. (Note: **Required** if role is `iam`We)

gce-only Parameters

The following parameters are only valid when the role is of type `"gce"`:

- `bound_zones` - (Optional) The list of zones that a GCE instance must belong to in order to be authenticated. If bound_instance_groups is provided, it is assumed to be a zonal group and the group must belong to this zone.

- `bound_regions` - (Optional) The list of regions that a GCE instance must belong to in order to be authenticated. If bound_instance_groups is provided, it is assumed to be a regional group and the group must belong to this region. If bound_zones are provided, this attribute is ignored.
- `bound_instance_groups` - (Optional) The instance groups that an authorized instance must belong to in order to be authenticated. If specified, either `bound_zones` or `bound_regions` must be set too.
- `bound_labels` - (Optional) A comma-separated list of GCP labels formatted as `"key:value"` strings that must be set on authorized GCE instances. Because GCP labels are not currently ACL'd, we recommend that this be used in conjunction with other restrictions.

For more details on the usage of each argument consult the Vault GCP API documentation (https://www.vaultproject.io/api/auth/gcp/index.html).

Attribute Reference

No additional attributes are exposed by this resource.

vault_gcp_secret_backend

Creates an GCP Secret Backend for Vault. GCP secret backends can then issue GCP OAuth token or Service Account keys, once a role has been added to the backend.

Important All data provided in the resource configuration will be written in cleartext to state and plan files generated by Terraform, and will appear in the console output when Terraform runs. Protect these artifacts accordingly. See the main provider documentation (/docs/providers/vault/index.html) for more details.

Example Usage

```
resource "vault_gcp_secret_backend" "gcp" {
  credentials = "${file("credentials.json")}
}
```

Argument Reference

The following arguments are supported:

- `credentials` - (Optional) The GCP service account credentails in JSON format.

Important Because Vault does not support reading the configured credentials back from the API, Terraform cannot detect and correct drift on `credentials`. Changing the values, however, *will* overwrite the previously stored values.

- `path` - (Optional) The unique path this backend should be mounted at. Must not begin or end with a `/`. Defaults to `gcp`.
- `description` - (Optional) A human-friendly description for this backend.
- `default_lease_ttl_seconds` - (Optional) The default TTL for credentials issued by this backend. Defaults to '3600'.
- `max_lease_ttl_seconds` - (Optional) The maximum TTL that can be requested for credentials issued by this backend. Defaults to '86400'.

Attributes Reference

No additional attributes are exported by this resource.

vault_generic_secret

Writes and manages arbitrary data at a given path in Vault.

This resource is primarily intended to be used with Vault's "generic" secret backend (https://www.vaultproject.io/docs/secrets/generic/index.html), but it is also compatible with any other Vault endpoint that supports the `vault write` command to create and the `vault delete` command to delete.

Important All data provided in the resource configuration will be written in cleartext to state and plan files generated by Terraform, and will appear in the console output when Terraform runs. Protect these artifacts accordingly. See the main provider documentation (/docs/providers/vault/index.html) for more details.

Example Usage

```
resource "vault_generic_secret" "example" {
  path = "secret/foo"

  data_json = <<EOT
{
  "foo":   "bar",
  "pizza": "cheese"
}
EOT
}
```

Argument Reference

The following arguments are supported:

- `path` - (Required) The full logical path at which to write the given data. To write data into the "generic" secret backend mounted in Vault by default, this should be prefixed with `secret/`. Writing to other backends with this resource is possible; consult each backend's documentation to see which endpoints support the `PUT` and `DELETE` methods.
- `data_json` - (Required) String containing a JSON-encoded object that will be written as the secret data at the given path.
- `allow_read` - (Optional, Deprecated) True/false. Set this to true if your vault authentication is able to read the data, this allows the resource to be compared and updated. Defaults to false.
- `disable_read` - (Optional) True/false. Set this to true if your vault authentication is not able to read the data. Setting this to `true` will break drift detection. Defaults to false.

Required Vault Capabilities

Use of this resource requires the `create` or `update` capability (depending on whether the resource already exists) on the given path, along with the `delete` capbility if the resource is removed from configuration.

This resource does not *read* the secret data back from Terraform on refresh by default. This avoids the need for `read` access on the given path, but it means that Terraform is not able to detect and repair "drift" on this resource should the data be updated or deleted outside of Terraform. This limitation can be negated by setting `allow_read` to true

Attributes Reference

No additional attributes are exported by this resource.

Import

Generic secrets can be imported using the `path`, e.g.

```
$ terraform import vault_mount.example secret/foo
```

vault_jwt_auth_backend_role

Manages an JWT auth backend role in a Vault server. See the Vault documentation (https://www.vaultproject.io/docs/auth/jwt.html) for more information.

Example Usage

```
resource "vault_auth_backend" "jwt" {
  type = "jwt"
}

resource "vault_jwt_auth_backend_role" "example" {
  backend   = "${vault_auth_backend.jwt.path}"
  role_name = "test-role"
  policies  = ["default", "dev", "prod"]

  bound_audiences = ["https://myco.test"]
  user_claim      = "https://vault/user"
}
```

Argument Reference

The following arguments are supported:

- `role_name` - (Required) The name of the role.
- `bound_audiences` - (Required) List of `aud` claims to match against. Any match is sufficient.
- `user_claim` - (Required) The claim to use to uniquely identify the user; this will be used as the name for the Identity entity alias created due to a successful login.
- `policies` - (Optional) Policies to be set on tokens issued using this role.
- `ttl` - (Optional) The initial/renewal TTL of tokens issued using this role, in seconds.
- `max_ttl` - (Optional) The maximum allowed lifetime of tokens issued using this role, in seconds.
- `period` - (Optional) If set, indicates that the token generated using this role should never expire, but instead always use the value set here as the TTL for every renewal.
- `num_uses` - (Optional) If set, puts a use-count limitation on the issued token.
- `bound_subject` - (Optional) If set, requires that the `sub` claim matches this value.
- `bound_cidrs` - (Optional) If set, a list of CIDRs valid as the source address for login requests. This value is also encoded into any resulting token.
- `groups_claim` - (Optional) The claim to use to uniquely identify the set of groups to which the user belongs; this will be used as the names for the Identity group aliases created due to a successful login. The claim value must be a list of strings.
- `backend` - (Optional) The unique name of the auth backend to configure. Defaults to `jwt`.

Attributes Reference

No additional attributes are exported by this resource.

Import

JWT authentication backend roles can be imported using the `path`, e.g.

```
$ terraform import vault_jwt_auth_backend_role.example auth/jwt/role/test-role
```

vault_kubernetes_auth_backend_config

Manages an Kubernetes auth backend config in a Vault server. See the Vault documentation (https://www.vaultproject.io/docs/auth/kubernetes.html) for more information.

Example Usage

```
resource "vault_auth_backend" "kubernetes" {
  type = "kubernetes"
}

resource "vault_kubernetes_auth_backend_config" "example" {
  backend            = "${vault_auth_backend.kubernetes.path}"
  kubernetes_host    = "http://example.com:443"
  kubernetes_ca_cert = "-----BEGIN CERTIFICATE-----\nexample\n-----END CERTIFICATE-----"
  token_reviewer_jwt = "ZXhhbXBsZQo="
}
```

Argument Reference

The following arguments are supported:

- `kubernetes_host` - (Required) Host must be a host string, a host:port pair, or a URL to the base of the Kubernetes API server.
- `kubernetes_ca_cert` - (Optional) PEM encoded CA cert for use by the TLS client used to talk with the Kubernetes API.
- `token_reviewer_jwt` - (Optional) A service account JWT used to access the TokenReview API to validate other JWTs during login. If not set the JWT used for login will be used to access the API.
- `pem_keys` - (Optional) List of PEM-formatted public keys or certificates used to verify the signatures of Kubernetes service account JWTs. If a certificate is given, its public key will be extracted. Not every installation of Kubernetes exposes these keys.

Attributes Reference

No additional attributes are exported by this resource.

vault_kubernetes_auth_backend_role

Manages an Kubernetes auth backend role in a Vault server. See the Vault documentation (https://www.vaultproject.io/docs/auth/kubernetes.html) for more information.

Example Usage

```
resource "vault_auth_backend" "kubernetes" {
  type = "kubernetes"
}

resource "vault_kubernetes_auth_backend_role" "example" {
  backend                          = "${vault_auth_backend.kubernetes.path}"
  role_name                        = "example-role"
  bound_service_account_names      = ["example"]
  bound_service_account_namespaces = ["example"]
  ttl                              = 3600
  policies                         = ["default", "dev", "prod"]
}
```

Argument Reference

The following arguments are supported:

- `role_name` - (Required) Name of the role.
- `bound_service_account_names` - (Optional) List of service account names able to access this role. If set to "*" all names are allowed, both this and bound_service_account_namespaces can not be "*".
- `bound_service_account_namespaces` - (Optional) List of namespaces allowed to access this role. If set to "*" all namespaces are allowed, both this and bound_service_account_names can not be set to "*".
- `ttl` - (Optional) The TTL period of tokens issued using this role in seconds.
- `max_ttl` - (Optional) The maximum allowed lifetime of tokens issued in seconds using this role.
- `period` - (Optional) If set, indicates that the token generated using this role should never expire. The token should be renewed within the duration specified by this value. At each renewal, the token's TTL will be set to the value of this parameter.
- `policies` - (Optional) Policies to be set on tokens issued using this role.
- `backend` - (Optional) Unique name of the kubernetes backend to configure.

Attributes Reference

No additional attributes are exported by this resource.

vault_ldap_auth_backend

Provides a resource for managing an LDAP auth backend within Vault (https://www.vaultproject.io/docs/auth/ldap.html).

Example Usage

```
resource "vault_ldap_auth_backend" "ldap" {
    path        = "ldap"
    url         = "ldaps://dc-01.example.org"
    userdn      = "OU=Users,OU=Accounts,DC=example,DC=org"
    userattr    = "sAMAccountName"
    upndomain   = "EXAMPLE.ORG"
    discoverdn  = false
    groupdn     = "OU=Groups,DC=example,DC=org"
    groupfilter = "(&(objectClass=group)(member:1.2.840.113556.1.4.1941:={{.UserDN}}))"
}
```

Argument Reference

The following arguments are supported:

- `url` - (Required) The URL of the LDAP server
- `starttls` - (Optional) Control use of TLS when conecting to LDAP
- `tls_min_version` - (Optional) Minimum acceptable version of TLS
- `tls_max_version` - (Optional) Maximum acceptable version of TLS
- `insecure_tls` - (Optional) Control whether or TLS certificates must be validated
- `certificate` - (Optional) Trusted CA to validate TLS certificate
- `binddn` - (Optional) DN of object to bind when performing user search
- `bindpass` - (Optional) Password to use with `binddn` when performing user search
- `userdn` - (Optional) Base DN under which to perform user search
- `userattr` - (Optional) Attribute on user object matching username passed in
- `upndomain` - (Optional) The userPrincipalDomain used to construct UPN string
- `discoverdn`: (Optional) Use anonymous bind to discover the bind DN of a user.
- `deny_null_bind`: (Optional) Prevents users from bypassing authentication when providing an empty password.
- `upndomain`: (Optional) The `userPrincipalDomain` used to construct the UPN string for the authenticating user.
- `groupfilter` - (Optional) Go template used to construct group membership query
- `groupdn` - (Optional) Base DN under which to perform group search
- `groupattr` - (Optional) LDAP attribute to follow on objects returned by groupfilter

- `path` - (Optional) Path to mount the LDAP auth backend under
- `description` - (Optional) Description for the LDAP auth backend mount

For more details on the usage of each argument consult the Vault LDAP API documentation (https://www.vaultproject.io/api/auth/ldap/index.html).

> **Important** Because Vault does not support reading the configured credentials back from the API, Terraform cannot detect and correct drift on `bindpass`. Changing the values, however, *will* overwrite the previously stored values.

Attributes Reference

In addition to the fields above, the following attributes are exported:

- `accessor` - The accessor for this auth mount.

vault_ldap_auth_backend_group

Provides a resource to create a group in an LDAP auth backend within Vault (https://www.vaultproject.io/docs/auth/ldap.html).

Example Usage

```
resource "vault_ldap_auth_backend" "ldap" {
    path        = "ldap"
    url         = "ldaps://dc-01.example.org"
    userdn      = "OU=Users,OU=Accounts,DC=example,DC=org"
    userattr    = "sAMAccountName"
    upndomain   = "EXAMPLE.ORG"
    discoverdn  = false
    groupdn     = "OU=Groups,DC=example,DC=org"
    groupfilter = "(&(objectClass=group)(member:1.2.840.113556.1.4.1941:={{.UserDN}}))"
}

resource "vault_ldap_auth_backend_group" "group" {
    groupname = "dba"
    policies  = ["dba"]
    backend   = "${vault_ldap_auth_backend.ldap.path}"
}
```

Argument Reference

The following arguments are supported:

- `groupname` - (Required) The LDAP groupname
- `policies` - (Optional) Policies which should be granted to members of the group
- `backend` - (Optional) Path to the authentication backend

For more details on the usage of each argument consult the Vault LDAP API documentation (https://www.vaultproject.io/api/auth/ldap/index.html).

Attribute Reference

No additional attributes are exposed by this resource.

vault_ldap_auth_backend_user

Provides a resource to create a user in an LDAP auth backend within Vault (https://www.vaultproject.io/docs/auth/ldap.html).

Example Usage

```
resource "vault_ldap_auth_backend" "ldap" {
    path        = "ldap"
    url         = "ldaps://dc-01.example.org"
    userdn      = "OU=Users,OU=Accounts,DC=example,DC=org"
    userattr    = "sAMAccountName"
    upndomain   = "EXAMPLE.ORG"
    discoverdn  = false
    groupdn     = "OU=Groups,DC=example,DC=org"
    groupfilter = "(&(objectClass=group)(member:1.2.840.113556.1.4.1941:={{.UserDN}}))"
}

resource "vault_ldap_auth_backend_user" "user" {
    username = "test-user"
    policies = ["dba", "sysops"]
    backend  = "${vault_ldap_auth_backend.ldap.path}"
}
```

Argument Reference

The following arguments are supported:

- `username` - (Required) The LDAP username
- `policies` - (Optional) Policies which should be granted to user
- `groups` - (Optional) Override LDAP groups which should be granted to user
- `backend` - (Optional) Path to the authentication backend

For more details on the usage of each argument consult the Vault LDAP API documentation (https://www.vaultproject.io/api/auth/ldap/index.html).

Attribute Reference

No additional attributes are exposed by this resource.

vault_mount

Example Usage

```
resource "vault_mount" "example" {
  path        = "dummy"
  type        = "generic"
  description = "This is an example mount"
}
```

Argument Reference

The following arguments are supported:

- `path` - (Required) Where the secret backend will be mounted
- `type` - (Required) Type of the backend, such as "aws"
- `description` - (Optional) Human-friendly description of the mount
- `default_lease_ttl_seconds` - (Optional) Default lease duration for tokens and secrets in seconds
- `max_lease_ttl_seconds` - (Optional) Maximum possible lease duration for tokens and secrets in seconds
- `options` - (Optional) Specifies mount type specific options that are passed to the backend

Attributes Reference

In addition to the fields above, the following attributes are exported:

- `accessor` - The accessor for this mount.

Import

Mounts can be imported using the `path`, e.g.

```
$ terraform import vault_mount.example dummy
```

vault_okta_auth_backend

Provides a resource for managing an Okta auth backend within Vault (https://www.vaultproject.io/docs/auth/okta.html).

Example Usage

```
resource "vault_okta_auth_backend" "example" {
    description  = "Demonstration of the Terraform Okta auth backend"
    organization = "example"
    token        = "something that should be kept secret"

    group {
        group_name = "foo"
        policies   = ["one", "two"]
    }

    user {
        username = "bar"
        groups   = ["foo"]
    }
}
```

Argument Reference

The following arguments are supported:

- `path` - (Required) Path to mount the Okta auth backend
- `description` - (Optional) The description of the auth backend
- `organization` - (Required) The Okta organization. This will be the first part of the url `https://XXX.okta.com`
- `token` - (Optional) The Okta API token. This is required to query Okta for user group membership. If this is not supplied only locally configured groups will be enabled.
- `base_url` - (Optional) The Okta url. Examples: oktapreview.com, okta.com
- `bypass_okta_mfa` - (Optional) When true, requests by Okta for a MFA check will be bypassed. This also disallows certain status checks on the account, such as whether the password is expired.
- `ttl` - (Optional) Duration after which authentication will be expired. See the documentation for info on valid duration formats (https://golang.org/pkg/time/#ParseDuration).
- `max_ttl` - (Optional) Maximum duration after which authentication will be expired See the documentation for info on valid duration formats (https://golang.org/pkg/time/#ParseDuration).
- `group` - (Optional) Associate Okta groups with policies within Vault. See below for more details.
- `user` - (Optional) Associate Okta users with groups or policies within Vault. See below for more details.

Okta Group

- `group_name` - (Required) Name of the group within the Okta
- `policies` - (Optional) Vault policies to associate with this group

Okta User

- `username` - (Required Optional) Name of the user within Okta
- `groups` - (Optional) List of Okta groups to associate with this user
- `policies` - (Optional) List of Vault policies to associate with this user

Attributes Reference

No additional attributes are exposed by this resource.

vault_okta_auth_backend_group

Provides a resource to create a group in an Okta auth backend within Vault (https://www.vaultproject.io/docs/auth/okta.html).

Example Usage

```
resource "vault_okta_auth_backend" "example" {
    path         = "group_okta"
    organization = "dummy"
}

resource "vault_okta_auth_backend_group" "foo" {
    path       = "${vault_okta_auth_backend.example.path}"
    group_name = "foo"
    policies   = ["one", "two"]
}
```

Argument Reference

The following arguments are supported:

- `path` - (Required) The path where the Okta auth backend is mounted
- `group_name` - (Required) Name of the group within the Okta
- `policies` - (Optional) Vault policies to associate with this group

Attributes Reference

No additional attributes are exposed by this resource.

vault_okta_auth_backend_user

Provides a resource to create a user in an Okta auth backend within Vault (https://www.vaultproject.io/docs/auth/okta.html).

Example Usage

```
resource "vault_okta_auth_backend" "example" {
    path         = "user_okta"
    organization = "dummy"
}

resource "vault_okta_auth_backend_user" "foo" {
    path     = "${vault_okta_auth_backend.example.path}"
    username = "foo"
    groups   = ["one", "two"]
}
```

Argument Reference

The following arguments are supported:

- `path` - (Required) The path where the Okta auth backend is mounted
- `username` - (Required Optional) Name of the user within Okta
- `groups` - (Optional) List of Okta groups to associate with this user
- `policies` - (Optional) List of Vault policies to associate with this user

Attributes Reference

No additional attributes are exposed by this resource.

vault_policy

Example Usage

```
resource "vault_policy" "example" {
  name = "dev-team"

  policy = <<EOT
path "secret/my_app" {
  policy = "write"
}
EOT
}
```

Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the policy
- `policy` - (Required) String containing a Vault policy

Attributes Reference

No additional attributes are exported by this resource.

Import

Policies can be imported using the `name`, e.g.

```
$ terraform import vault_policy.example dev-team
```

vault_rabbitmq_secret_backend

Creates an RabbitMQ Secret Backend for Vault. RabbitMQ secret backends can then issue RabbitMQ credentials, once a role has been added to the backend.

Important All data provided in the resource configuration will be written in cleartext to state and plan files generated by Terraform, and will appear in the console output when Terraform runs. Protect these artifacts accordingly. See the main provider documentation (/docs/providers/vault/index.html) for more details.

Example Usage

```
resource "vault_rabbitmq_secret_backend" "rabbitmq" {
  connection_uri = "https://....."
  username = "user"
  password = "password"
}
```

Argument Reference

The following arguments are supported:

- `connection_uri` - (Required) Specifies the RabbitMQ connection URI.
- `username` - (Required) Specifies the RabbitMQ management administrator username.
- `password` - (Required) Specifies the RabbitMQ management administrator password.
- `verify_connection` - (Optional) Specifies whether to verify connection URI, username, and password. Defaults to `true`.

Important Because Vault does not support reading the configured credentials back from the API, Terraform cannot detect and correct drift on `connection_uri`, `username`, `password` or `verify_connection`. Changing the values, however, *will* overwrite the previously stored values.

- `path` - (Optional) The unique path this backend should be mounted at. Must not begin or end with a `/`. Defaults to `aws`.
- `description` - (Optional) A human-friendly description for this backend.
- `default_lease_ttl_seconds` - (Optional) The default TTL for credentials issued by this backend.
- `max_lease_ttl_seconds` - (Optional) The maximum TTL that can be requested for credentials issued by this backend.

Attributes Reference

No additional attributes are exported by this resource.

Import

RabbitMQ secret backends can be imported using the `path`, e.g.

```
$ terraform import vault_rabbitmq_secret_backend.rabbitmq rabbitmq
```

vault_rabbitmq_secret_backend_role

Creates a role on an RabbitMQ Secret Backend for Vault. Roles are used to map credentials to the policies that generated them.

Important All data provided in the resource configuration will be written in cleartext to state and plan files generated by Terraform, and will appear in the console output when Terraform runs. Protect these artifacts accordingly. See the main provider documentation (/docs/providers/vault/index.html) for more details.

Example Usage

```
resource "vault_rabbitmq_secret_backend" "rabbitmq" {
  connection_uri = "https://....."
  username = "user"
  password = "password"
}

resource "vault_rabbitmq_secret_backend_role" "role" {
  backend = "${vault_rabbitmq_secret_backend.rabbitmq.path}"
  name    = "deploy"

  tags = "tag1,tag2"
  vhost = "{\"/\": {\"configure\":\".*\", \"write\":\".*\", \"read\": \".*\"}}"
}
```

Argument Reference

The following arguments are supported:

- `backend` - (Required) The path the RabbitMQ secret backend is mounted at, with no leading or trailing /s.
- `name` - (Required) The name to identify this role within the backend. Must be unique within the backend.
- `tags` - (Optional) Specifies a comma-separated RabbitMQ management tags.
- `vhost` - (Optional) Specifies a map of virtual hosts to permissions.

Attributes Reference

No additional attributes are exported by this resource.

Import

RabbitMQ secret backend roles can be imported using the `path`, e.g.

```
$ terraform import vault_rabbitmq_secret_backend_role.role rabbitmq/roles/deploy
```

vault_ssh_secret_backend_ca

Provides a resource to manage CA information in an SSH secret backend SSH secret backend within Vault (https://www.vaultproject.io/docs/secrets/ssh/index.html).

Example Usage

```
resource "vault_mount" "example" {
    type = "ssh"
}

resource "vault_ssh_secret_backend_ca" "foo" {
    backend = "${vault_mount.example.path}"
}
```

Argument Reference

The following arguments are supported:

- `backend` - (Optional) The path where the SSH secret backend is mounted. Defaults to 'ssh'
- `generate_signing_key` - (Optional) Whether Vault should generate the signing key pair internally. Defaults to true
- `public_key` - (Optional) The public key part the SSH CA key pair; required if generate_signing_key is false.
- `private_key` - (Optional) The private key part the SSH CA key pair; required if generate_signing_key is false.

Important Because Vault does not support reading the private_key back from the API, Terraform cannot detect and correct drift on `private_key`. Changing the values, however, *will* overwrite the previously stored values.

Attributes Reference

No additional attributes are exposed by this resource.

vault_token_auth_backend_role

Manages Token auth backend role in a Vault server. See the Vault documentation (https://www.vaultproject.io/docs/auth/token.html) for more information.

Example Usage

```
resource "vault_token_auth_backend_role" "example" {
  role_name           = "my-role"
  allowed_policies    = ["dev", "test"]
  disallowed_policies = ["default"]
  orphan              = true
  period              = "86400"
  renewable           = true
  explicit_max_ttl    = "115200"
  path_suffix         = "path-suffix"
}
```

Argument Reference

The following arguments are supported:

- `role_name` - (Required) The name of the role.
- `allowed_policies` (Optional) List of allowed policies for given role.
- `disallowed_policies` (Optional) List of disallowed policies for given role.
- `orphan` (Optional) If true, tokens created against this policy will be orphan tokens.
- `period` (Optional) The duration in which a token should be renewed. At each renewal, the token's TTL will be set to the value of this parameter.
- `renewable` (Optional) Wether to disable the ability of the token to be renewed past its initial TTL.
- `explicit_max_ttl` (Optional) If set, the token will have an explicit max TTL set upon it.
- `path_suffix` (Optional) Tokens created against this role will have the given suffix as part of their path in addition to the role name.
- `ttl` (Optional) The TTL period of tokens issued using this role, provided as the number of minutes.
- `max_ttl` (Optional) The maximum allowed lifetime of tokens issued using this role.

Attributes Reference

No additional attributes are exported by this resource.

Import

Token auth backend roles can be imported with `auth/token/roles/` followed by the `role_name`, e.g.

```
$ terraform import vault_token_auth_backend_role.example auth/token/roles/my-role
```