» acme_registration

The `acme_registration` resource can be used to create and manage accounts on an ACME server. Once registered, the same private key that has been used for registration can be used to request authorizations for certificates.

This resource is named `acme_registration` for historical reasons - in the ACME v1 spec, a *registration* referred to the account entity. This resource name is stable and more than likely will not change until a later major version of the provider, if at all.

Keep in mind that when using this resource along with `acme_certificate` within the same configuration, a change in the provider-level `server_url` (example: from the Let's Encrypt staging to production environment) within the same Terraform state will result in a resource failure, as Terraform will attempt to look for the account in the wrong CA. Consider different workspaces per environment, and/or using multiple provider instances.

» Example

The following creates an account off of a private key generated with the `tls_private_key` resource.

```
provider "acme" {
  server_url = "https://acme-staging-v02.api.letsencrypt.org/directory"
}

resource "tls_private_key" "private_key" {
  algorithm = "RSA"
}

resource "acme_registration" "reg" {
  account_key_pem = "${tls_private_key.private_key.private_key_pem}"
  email_address   = "nobody@example.com"
}
```

» Argument Reference

NOTE: All arguments in `acme_registration` force a new resource if changed.

The resource takes the following arguments:

- `account_key_pem` (Required) - The private key used to identity the account.
- `email_address` (Required) - The contact email address for the account.

» Attribute Reference

The following attributes are exported:

- `id`: The original full URL of the account.
- `registration_url`: The current full URL of the account.

`id` and `registration_url` will usually be the same and will usually only diverge when migrating protocols, ie: ACME v1 to v2.

» acme_certificate

The `acme_certificate` resource can be used to create and manage an ACME TLS certificate.

NOTE: As the usage model of Terraform generally sees it as being run on a different server than a certificate would normally be placed on, the `acme_certificate` resource only supports DNS challenges.

» Example

The below example is the same example that can be found on the index page, and creates both an account and certificate within the same configuration. The account is created using the `acme_registration` resource.

NOTE: When creating accounts and certificates within the same configuration, ensure that you reference the `account_key_pem` argument in the `acme_registration` resource as the corresponding `account_key_pem` argument in the `acme_certificate` resource. This will ensure that the account gets created before the certificate and avoid errors.

```
provider "acme" {
  server_url = "https://acme-staging-v02.api.letsencrypt.org/directory"
}

resource "tls_private_key" "private_key" {
  algorithm = "RSA"
}

resource "acme_registration" "reg" {
  account_key_pem = "${tls_private_key.private_key.private_key_pem}"
  email_address   = "nobody@example.com"
}

resource "acme_certificate" "certificate" {
  account_key_pem           = "${acme_registration.reg.account_key_pem}"
  common_name               = "www.example.com"
  subject_alternative_names = ["www2.example.com"]

  dns_challenge {
    provider = "route53"
  }
}
```

» Using an external CSR

The `acme_certificate` resource can also take an external CSR. In this example, we create one using `tls_cert_request` first, before supplying it to the `certificate_request_pem` argument.

NOTE: Some current ACME CA implementations (including Let's Encrypt) strip most of the organization information out of a certificate request subject. You may wish to confirm with the CA what behavior to expect when using the `certificate_request_pem` argument with this resource.

NOTE: It is not a good practice to use the same private key for both your account and your certificate. Make sure you use different keys.

```
provider "acme" {
  server_url = "https://acme-staging-v02.api.letsencrypt.org/directory"
}

resource "tls_private_key" "reg_private_key" {
  algorithm = "RSA"
}

resource "acme_registration" "reg" {
  account_key_pem = "${tls_private_key.reg_private_key.private_key_pem}"
  email_address   = "nobody@example.com"
}

resource "tls_private_key" "cert_private_key" {
  algorithm = "RSA"
}

resource "tls_cert_request" "req" {
  key_algorithm   = "RSA"
  private_key_pem = "${tls_private_key.cert_private_key.private_key_pem}"
  dns_names       = ["www.example.com", "www2.example.com"]

  subject {
    common_name = "www.example.com"
  }
}

resource "acme_certificate" "certificate" {
  account_key_pem         = "${acme_registration.reg.account_key_pem}"
  certificate_request_pem = "${tls_cert_request.req.cert_request_pem}"

  dns_challenge {
    provider = "route53"
  }
}
```

» Argument Reference

The resource takes the following arguments:

NOTE: All arguments in `acme_certificate`, other than `min_days_remaining`, force a new resource when changed.

- `account_key_pem` (Required) - The private key of the account that is requesting the certificate.
- `common_name` - The certificate's common name, the primary domain that the certificate will be recognized for. Required when not specifying a CSR.
- `subject_alternative_names` - The certificate's subject alternative names, domains that this certificate will also be recognized for. Only valid when not specifying a CSR.
- `key_type` - The key type for the certificate's private key. Can be one of: `P256` and `P384` (for ECDSA keys of respective length) or `2048`, `4096`, and `8192` (for RSA keys of respective length). Required when not specifying a CSR. The default is `2048` (RSA key of 2048 bits).
- `certificate_request_pem` - A pre-created certificate request, such as one from `tls_cert_request`, or one from an external source, in PEM format. Either this, or the in-resource request options (`common_name`, `key_type`, and optionally `subject_alternative_names`) need to be specified.
- `dns_challenge` (Required) - The DNS challenges to use in fulfilling the request.
- `recursive_nameservers` (Optional) - The recursive nameservers that will be used to check for propagation of the challenge record. Defaults to your system-configured DNS resolvers.
- `must_staple` (Optional) Enables the OCSP Stapling Required TLS Security Policy extension. Certificates with this extension must include a valid OCSP Staple in the TLS handshake for the connection to succeed. Defaults to `false`. Note that this option has no effect when using an external CSR - it must be enabled in the CSR itself.

NOTE: OCSP stapling requires specific webserver configuration to support the downloading of the staple from the CA's OCSP endpoints, and should be configured to tolerate prolonged outages of the OCSP service. Consider this when using `must_staple`, and only enable it if you are sure your webserver or service provider can be configured correctly.

- `min_days_remaining` (Optional) - The minimum amount of days remaining on the expiration of a certificate before a renewal is attempted. The default is `30`. A value of less than `0` means that the certificate will never be renewed.
- `certificate_p12_password` - (Optional) Password to be used when generating the PFX file stored in `certificate_p12`. Defaults to an empty string.

» Using DNS challenges

As the usage model of Terraform generally sees it as being run on a different server than a certificate would normally be placed on, the `acme_certificate` resource only supports DNS challenges. This method authenticates certificate domains by requiring the requester to place a TXT record on the FQDNs in the certificate.

The ACME provider responds to DNS challenges automatically by utilizing one of the supported DNS challenge providers. Most providers take credentials as environment variables, but if you would rather use configuration for this purpose, you can by specifying `config` blocks within a `dns_challenge` block, along with the `provider` parameter.

For a full list of providers, click here.

Example with the Route 53 provider:

```
resource "acme_certificate" "certificate" {
  #...

  dns_challenge {
    provider = "route53"

    config = {
      AWS_ACCESS_KEY_ID     = "${var.aws_access_key}"
      AWS_SECRET_ACCESS_KEY = "${var.aws_secret_key}"
      AWS_DEFAULT_REGION    = "us-east-1"
    }
  }

  #...
}
```

» Using Variable Files for Provider Arguments

Most provider arguments can be suffixed with `_FILE` to specify that you wish to store that value in a local file. This can be useful if local storage for these values is desired over configuration as variables or within the environment.

Building on the above Route 53 provider example, the following example uses local files to get the access key ID and secret access key.

```
resource "acme_certificate" "certificate" {
  #...

  dns_challenge {
    provider = "route53"

    config = {
      AWS_ACCESS_KEY_ID_FILE     = "/data/secrets/aws_access_key_id"
      AWS_SECRET_ACCESS_KEY_FILE = "/data/secrets/aws_secret_access_key"
      AWS_DEFAULT_REGION         = "us-east-1"
    }
  }

  #...
}
```

» Manually specifying recursive nameservers for propagation checks

The ACME provider will normally use your system-configured DNS resolvers to check for propagation of the TXT records before proceeding with the certificate request. In split horizon scenarios, this check may never succeed, as the machine running Terraform may not have visibility into these public DNS records.

To override this default behavior, supply the `recursive_nameservers` to use as a list in `host:port` form within the `dns_challenge` block:

```
resource "acme_certificate" "certificate" {
  #...

  recursive_nameservers = ["8.8.8.8:53"]

  dns_challenge {
    provider = "route53"
  }

  #...
}
```

» Using multiple primary DNS providers

The ACME provider will allow you to configure multiple DNS challenges in the event that you have more than one primary DNS provider.

```
resource "acme_certificate" "certificate" {
  #...

  dns_challenge {
    provider = "azure"
  }

  dns_challenge {
    provider = "gcloud"
  }

  dns_challenge {
    provider = "route53"
  }

  #...
}
```

Some considerations need to be kept in mind when using multiple providers:

- You cannot use more than one provider of the same type at once.
- Your NS records must be correctly configured so that each DNS challenge provider can correctly discover the appropriate zone to update.
- DNS propagation checks are conducted once per configured common name and subject alternative name, using the highest configured or default propagation timeout (`*_PROPAGATION_TIMEOUT`) and polling interval (`*_POLLING_INTERVAL`) settings.

» Relation to Terraform provider configuration

The DNS provider configuration specified in the `acme_certificate` resource is separate from any that you supply in a corresponding provider whose functionality overlaps with the certificate's DNS providers. This ensures that there are no hard dependencies between any of these providers and the ACME provider, but it is important to note so that configuration is supplied correctly.

As an example, if you specify manual configuration for the AWS provider via the `provider` block instead of the environment, you will still need to supply the configuration explicitly as per above.

Some of these providers have environment variable settings that overlap with the ones found here, generally depending on whether or not these variables are supported by the corresponding provider's SDK.

Check the DNS provider page of a specific provider for more details on exactly what variables are supported.

» Certificate renewal

The `acme_certificate` resource handles automatic certificate renewal so long as a plan or apply is done within the number of days specified in the `min_days_remaining` resource parameter. During refresh, if Terraform detects that the certificate is within the expiry range specified in `min_days_remaining`, or is already expired, Terraform will mark the certificate to be renewed on the next apply.

Note that a value less than `0` supplied to `min_days_remaining` will cause renewal checks to be bypassed, and the certificate will never renew.

» Attribute Reference

The following attributes are exported:

- `id` - The full URL of the certificate within the ACME CA.
- `certificate_url` - The full URL of the certificate within the ACME CA. Same as `id`.
- `certificate_domain` - The common name of the certificate.
- `private_key_pem` - The certificate's private key, in PEM format, if the certificate was generated from scratch and not with `certificate_request_pem`. If `certificate_request_pem` was used, this will be blank.
- `certificate_pem` - The certificate in PEM format.
- `issuer_pem` - The intermediate certificate of the issuer.
- `certificate_p12` - The certificate, intermediate, and the private key archived as a PFX file (PKCS12 format, generally used by Microsoft products). The data is base64 encoded (including padding), and its password is configurable via the `certificate_p12_password` argument. This field is empty if creating a certificate from a CSR.