Azure Active Directory Provider

The Azure Provider can be used to configure infrastructure in Azure Active Directory (https://azure.microsoft.com/en-us/services/active-directory/) using the Azure Resource Manager API's. Documentation regarding the Data Sources (/docs/configuration/data-sources.html) and Resources (/docs/configuration/resources.html) supported by the Azure Active Directory Provider can be found in the navigation to the left.

Interested in the provider's latest features, or want to make sure you're up to date? Check out the changelog (https://github.com/terraform-providers/terraform-provider-azuread/blob/master/CHANGELOG.md) for version information and release notes.

Authenticating to Azure Active Directory

Terraform supports a number of different methods for authenticating to Azure Active Directory:

- Authenticating to Azure Active Directory using the Azure CLI (/docs/providers/azuread/auth/azure_cli.html)
- Authenticating to Azure Active Directory using Managed Service Identity (/docs/providers/azuread/auth/managed_service_identity.html)
- Authenticating to Azure Active Directory using a Service Principal and a Client Certificate (/docs/providers/azuread/auth/service_principal_client_certificate.html)
- Authenticating to Azure Active Directory using a Service Principal and a Client Secret (/docs/providers/azuread/auth/service_principal_client_secret.html)

We recommend using either a Service Principal or Managed Service Identity when running Terraform non-interactively (such as when running Terraform in a CI server) - and authenticating using the Azure CLI when running Terraform locally.

Example Usage

```
# Configure the Microsoft Azure Active Directory Provider
provider "azuread" {
  version = "=0.3.0"
}

# Create an application
resource "azuread_application" "example" {
  name = "ExampleApp"
}

# Create a service principal
resource "azuread_service_principal" "example" {
  application_id = "${azuread_application.example.application_id}"
}
```

Features and Bug Requests

The Azure Active Directory provider's bugs and feature requests can be found in the GitHub repo issues (https://github.com/terraform-providers/terraform-provider-azuread/issues). Please avoid "me too" or "+1" comments. Instead, use a thumbs up reaction (https://blog.github.com/2016-03-10-add-reactions-to-pull-requests-issues-and-comments/) on enhancement requests. Provider maintainers will often prioritize work based on the number of thumbs on an issue.

Community input is appreciated on outstanding issues! We love to hear what use cases you have for new features, and want to provide the best possible experience for you using the Azure Active Directory provider.

If you have a bug or feature request without an existing issue

- if an existing resource or field is working in an unexpected way, file a bug (https://github.com/terraform-providers/terraform-provider-azuread/issues/new?template=bug.md).
- if you'd like the provider to support a new resource or field, file an enhancement/feature request (https://github.com/terraform-providers/terraform-provider-azuread/issues/new?template=enhancement.md).

The provider maintainers will often use the assignee field on an issue to mark who is working on it.

- An issue assigned to an individual maintainer indicates that maintainer is working on the issue
- If you're interested in working on an issue please leave a comment in that issue

---

If you have configuration questions, or general questions about using the provider, try checking out:

- Terraform's community resources (https://www.terraform.io/docs/extend/community/index.html)
- HashiCorp support (https://support.hashicorp.com) for Terraform Enterprise customers

Argument Reference

The following arguments are supported:

- `client_id` - (Optional) The Client ID which should be used. This can also be sourced from the `ARM_CLIENT_ID` Environment Variable.
- `environment` - (Optional) The Cloud Environment which be used. Possible values are `public`, `usgovernment`, `german` and `china`. Defaults to `public`. This can also be sourced from the `ARM_ENVIRONMENT` environment variable.
- `subscription_id` - (Optional) The Subscription ID which should be used. This can also be sourced from the `ARM_SUBSCRIPTION_ID` Environment Variable.
- `tenant_id` - (Optional) The Tenant ID which should be used. This can also be sourced from the `ARM_TENANT_ID` Environment Variable.

When authenticating as a Service Principal using a Client Certificate, the following fields can be set:

- `client_certificate_password` - (Optional) The password associated with the Client Certificate. This can also be sourced from the `ARM_CLIENT_CERTIFICATE_PASSWORD` Environment Variable.

- `client_certificate_path` - (Optional) The path to the Client Certificate associated with the Service Principal which should be used. This can also be sourced from the `ARM_CLIENT_CERTIFICATE_PATH` Environment Variable.

More information on how to configure a Service Principal using a Client Certificate can be found in this guide (/docs/providers/azuread/auth/service_principal_client_certificate.html).

When authenticating as a Service Principal using a Client Secret, the following fields can be set:

- `client_secret` - (Optional) The Client Secret which should be used. This can also be sourced from the `ARM_CLIENT_SECRET` Environment Variable.

More information on how to configure a Service Principal using a Client Secret can be found in this guide (/docs/providers/azuread/auth/service_principal_client_secret.html).

When authenticating using Managed Service Identity, the following fields can be set:

- `msi_endpoint` - (Optional) The path to a custom endpoint for Managed Service Identity - in most circumstances this should be detected automatically. This can also be sourced from the `ARM_MSI_ENDPOINT` Environment Variable.
- `use_msi` - (Optional) Should Managed Service Identity be used for Authentication? This can also be sourced from the `ARM_USE_MSI` Environment Variable. Defaults to `false`.

More information on how to configure a Service Principal using Managed Service Identity can be found in this guide (/docs/providers/azuread/auth/managed_service_identity.html).

It's also possible to use multiple Provider blocks within a single Terraform configuration, for example to work with resources across multiple Azure Active Directory Environments - more information can be found in the documentation for Providers (https://www.terraform.io/docs/configuration/providers.html#multiple-provider-instances).

Azure Active Directory Provider: Authenticating using the Azure CLI

Terraform supports a number of different methods for authenticating to Azure:

- Authenticating to Azure using the Azure CLI (which is covered in this guide)
- Authenticating to Azure using Managed Service Identity (/docs/providers/azuread/auth/managed_service_identity.html)
- Authenticating to Azure using a Service Principal and a Client Certificate (/docs/providers/azuread/auth/service_principal_client_certificate.html)
- Authenticating to Azure using a Service Principal and a Client Secret (/docs/providers/azuread/auth/service_principal_client_secret.html)

We recommend using either a Service Principal or Managed Service Identity when running Terraform non-interactively (such as when running Terraform in a CI server) - and authenticating using the Azure CLI when running Terraform locally.

Important Notes about Authenticating using the Azure CLI

- Prior to version 1.20 the AzureAD Provider used a different method of authorizing via the Azure CLI where credentials reset after an hour - as such we'd recommend upgrading to version 1.20 or later of the AzureAD Provider.
- Terraform only supports authenticating using the `az` CLI (and this must be available on your PATH) - authenticating using the older `azure` CLI or PowerShell Cmdlets is not supported.
- Authenticating via the Azure CLI is only supported when using a User Account. If you're using a Service Principal (for example via `az login --service-principal`) you should instead authenticate via the Service Principal directly (either using a Client Secret (/docs/providers/azuread/auth/service_principal_client_secret.html) or a Client Certificate (/docs/providers/azuread/auth/service_principal_client_certificate.html)).

Logging into the Azure CLI

Note: If you're using the **China**, **German** or **Government** Azure Clouds - you'll need to first configure the Azure CLI to work with that Cloud. You can do this by running:

```
$ az cloud set --name AzureChinaCloud|AzureGermanCloud|AzureUSGovernment
```

Firstly, login to the Azure CLI using:

```
$ az login
```

Once logged in - it's possible to list the Subscriptions associated with the account via:

```
$ az account list
```

The output (similar to below) will display one or more Subscriptions - with the `id` field being the `subscription_id` field referenced above.

```
[
  {
    "cloudName": "AzureCloud",
    "id": "00000000-0000-0000-0000-000000000000",
    "isDefault": true,
    "name": "PAYG Subscription",
    "state": "Enabled",
    "tenantId": "00000000-0000-0000-0000-000000000000",
    "user": {
      "name": "user@example.com",
      "type": "user"
    }
  }
]
```

Should you have more than one Subscription, you can specify the Subscription to use via the following command:

```
$ az account set --subscription="SUBSCRIPTION_ID"
```

Configuring Azure CLI authentication in Terraform

Now that we're logged into the Azure CLI - we can configure Terraform to use these credentials.

To configure Terraform to use the Default Subscription defined in the Azure CLI - we can use the following Provider block:

```
provider "azuread" {
  # Whilst version is optional, we /strongly recommend/ using it to pin the version of the Provider being
 used
  version = "=0.1.0"
}
```

More information on the fields supported in the Provider block can be found here (/docs/providers/azuread/index.html#argument-reference).

At this point running either `terraform plan` or `terraform apply` should allow Terraform to run using the Azure CLI to authenticate.

It's also possible to configure Terraform to use a specific Subscription - for example:

```
provider "azuread" {
  # Whilst version is optional, we /strongly recommend/ using it to pin the version of the Provider being
 used
  version = "=0.1.0"

  subscription_id = "00000000-0000-0000-0000-000000000000"
}
```

More information on the fields supported in the Provider block can be found here (/docs/providers/azuread/index.html#argument-reference).

At this point running either `terraform plan` or `terraform apply` should allow Terraform to run using the Azure CLI to authenticate.

If you're looking to use Terraform across Tenants - it's possible to do this by configuring the Tenant ID field in the Provider block, as shown below:

```
provider "azuread" {
  # Whilst version is optional, we /strongly recommend/ using it to pin the version of the Provider being
 used
  version = "=0.1.0"

  subscription_id = "00000000-0000-0000-0000-000000000000"
  tenant_id       = "11111111-1111-1111-1111-111111111111"
}
```

More information on the fields supported in the Provider block can be found here (/docs/providers/azuread/index.html#argument-reference).

At this point running either `terraform plan` or `terraform apply` should allow Terraform to run using the Azure CLI to authenticate.

Azure Active Directory Provider: Authenticating using Managed Service Identity

Terraform supports a number of different methods for authenticating to Azure:

- Authenticating to Azure using the Azure CLI (/docs/providers/azuread/auth/azure_cli.html)
- Authenticating to Azure using Managed Service Identity (which is covered in this guide)
- Authenticating to Azure using a Service Principal and a Client Certificate (/docs/providers/azuread/auth/service_principal_client_certificate.html)
- Authenticating to Azure using a Service Principal and a Client Secret (/docs/providers/azuread/auth/service_principal_client_secret.html)

We recommend using either a Service Principal or Managed Service Identity when running Terraform non-interactively (such as when running Terraform in a CI server) - and authenticating using the Azure CLI when running Terraform locally.

What is Managed Service Identity?

Certain services within Azure (for example Virtual Machines and Virtual Machine Scale Sets) can be assigned an Azure Active Directory identity which can be used to access the Azure Subscription. This identity can then be assigned permissions to a Subscription, Resource Group or other resources using the Azure Identity and Access Management functionality - however by default no permissions are assigned.

Once a resource is configured with an identity, a local metadata service exposes credentials which can be used by applications such as Terraform.

Configuring Managed Service Identity

The (simplified) Terraform Configuration below configures a Virtual Machine with Managed Service Identity, and then grants it Contributor access to the Subscription:

```
data "azuread_subscription" "current" {}

resource "azuread_virtual_machine" "test" {
  # ...

  identity = {
    type = "SystemAssigned"
  }
}

data "azuread_builtin_role_definition" "contributor" {
  name = "Contributor"
}

resource "azuread_role_assignment" "test" {
  name               = "${azuread_virtual_machine.test.name}"
  scope              = "${data.azuread_subscription.primary.id}"
  role_definition_id = "${data.azuread_subscription.subscription.id}${data.azuread_builtin_role_definitio
n.contributor.id}"
  principal_id       = "${lookup(azuread_virtual_machine.test.identity[0], "principal_id")}"
}
```

Configuring Managed Service Identity in Terraform

At this point we assume that Managed Service Identity is configured on the resource (e.g. Virtual Machine) being used - and that permissions have been assigned via Azure's Identity and Access Management system.

Terraform can be configured to use Managed Service Identity for authentication in one of two ways: using Environment Variables or by defining the fields within the Provider block.

You can configure Terraform to use Managed Service Identity by setting the Environment Variable `ARM_USE_MSI` to `true`; as shown below:

```
$ export ARM_USE_MSI=true
```

Using a Custom MSI Endpoint? In the unlikely event you're using a custom endpoint for Managed Service Identity - this can be configured using the `ARM_MSI_ENDPOINT` Environment Variable - however this shouldn't need to be configured in regular use.

Whilst a Provider block is *technically* optional when using Environment Variables - we'd strongly recommend defining one to be able to pin the version of the Provider being used:

```
provider "azuread" {
  # Whilst version is optional, we /strongly recommend/ using it to pin the version of the Provider being
 used
  version = "=0.1.0"
}
```

More information on the fields supported in the Provider block can be found here (/docs/providers/azuread/index.html#argument-reference).

At this point running either `terraform plan` or `terraform apply` should allow Terraform to run using Managed Service Identity.

It's also possible to configure Managed Service Identity within the Provider Block:

```
provider "azuread" {
  # Whilst version is optional, we /strongly recommend/ using it to pin the version of the Provider being
 used
  version = "=0.1.0"

  use_msi = true
}
```

Using a Custom MSI Endpoint? In the unlikely event you're using a custom endpoint for Managed Service Identity - this can be configured using the `msi_endpoint` field - however this shouldn't need to be configured in regular use.

More information on the fields supported in the Provider block can be found here (/docs/providers/azuread/index.html#argument-reference).

At this point running either `terraform plan` or `terraform apply` should allow Terraform to run using Managed Service Identity.

Azure Active Directory Provider: Authenticating using a Service Principal with a Client Certificate

Terraform supports a number of different methods for authenticating to Azure:

- Authenticating to Azure using the Azure CLI (/docs/providers/azuread/auth/azure_cli.html)
- Authenticating to Azure using Managed Service Identity (/docs/providers/azuread/auth/managed_service_identity.html)
- Authenticating to Azure using a Service Principal and a Client Certificate (which is covered in this guide)
- Authenticating to Azure using a Service Principal and a Client Secret (/docs/providers/azuread/auth/service_principal_client_secret.html)

Further steps must be taken to grant a Service Principal permission to manage objects in an Azure Active Directory:

Granting a Service Principal permission to manage AAD (/docs/providers/azuread/auth/service_principal_configuration.html)

We recommend using either a Service Principal or Managed Service Identity when running Terraform non-interactively (such as when running Terraform in a CI server) - and authenticating using the Azure CLI when running Terraform locally.

Beyond authentication and managing Azure AAD resources further steps are required to make so a Service principal can make changes to Azure Active Directory objects such as users and groups. The Granting a Service Principal permission to manage AAD (/docs/providers/azuread/auth/service_principal_configuration.html) guide contains the required steps.

Creating a Service Principal

A Service Principal is an application within Azure Active Directory which can be used as a means of authentication, either using a Client Secret (/docs/providers/azuread/auth/service_principal_client_secret.html) or a Client Certificate (which is documented in this guide) and can be created though the Azure Portal.

This guide will cover how to generate a client certificate, how to create a Service Principal and then how to assign the Client Certificate to the Service Principal so that it can be used for authentication. Once that's done finally we're going to grant the Service Principal permission to manage resources in the Subscription - to do this we're going to assign `Contributor` rights to the Subscription - however it's possible to assign other permissions (https://azure.microsoft.com/en-gb/documentation/articles/role-based-access-built-in-roles/) depending on your configuration.

Generating a Client Certificate

Firstly we need to create a certificate which can be used for authentication. To do that we're going to generate a Certificate Signing Request (also known as a CSR) using `openssl` (this can also be achieved using PowerShell, however that's outside the scope of this document):

```
$ openssl req -newkey rsa:4096 -nodes -keyout "service-principal.key" -out "service-principal.csr"
```

During the generation of the certificate you'll be prompted for various bits of information required for the certificate signing request - at least one item has to be specified for this to complete.

We can now sign that Certificate Signing Request, in this example we're going to self-sign this certificate using the Key we just generated; however it's also possible to do this using a Certificate Authority. In order to do that we're again going to use `openssl`:

```
$ openssl x509 -signkey "service-principal.key" -in "service-principal.csr" -req -days 365 -out "service-principal.crt"
```

Finally we can generate a PFX file which can be used to authenticate with Azure:

```
$ openssl pkcs12 -export -out "service-principal.pfx" -inkey "service-principal.key" -in "service-principal.crt"
```

Now that we've generated a certificate, we can create the Azure Active Directory application.

Creating the Service Principal

We're going to create the Service Principal in the Azure Portal - to do this navigate to the **Azure Active Directory** overview (https://portal.azure.com/#blade/Microsoft_AAD_IAM/ActiveDirectoryMenuBlade/Overview) within the Azure Portal - then select the **App Registration** blade (https://portal.azure.com/#blade/Microsoft_AAD_IAM/ActiveDirectoryMenuBlade/RegisteredApps/RegisteredApps/Overview) and click **Endpoints** at the top of the **App Registration** blade. A list of URIs will be displayed and you need to locate the URI for **OAUTH 2.0 AUTHORIZATION ENDPOINT** which contains a GUID. This GUID is your Tenant ID (the `tenant_id` field mentioned above).

Next, navigate back to the **App Registration** blade (https://portal.azure.com/#blade/Microsoft_AAD_IAM/ActiveDirectoryMenuBlade/RegisteredApps/RegisteredApps/Overview) - from here we'll create the Application in Azure Active Directory. To do this click **New application registration** at the top to add a new Application within Azure Active Directory. On this page, set the following values then press **Create**:

- **Name** - this is a friendly identifier and can be anything (e.g. "Terraform")
- **Application Type** - this should be set to "Web app / API"
- **Sign-on URL** - this can be anything, providing it's a valid URI (e.g. https://terra.form (https://terra.form))

At this point the newly created Azure Active Directory application should be visible on-screen - if it's not, navigate to the the **App Registration** blade (https://portal.azure.com/#blade/Microsoft_AAD_IAM/ActiveDirectoryMenuBlade/RegisteredApps/RegisteredApps/Overview) and select the Azure Active Directory application. At the top of this page, the "Application ID" GUID is the `client_id` you'll need.

Assigning the Client Certificate to the Service Principal

To associate the public portion of the Client Certificate (the `*.crt` file) with the Azure Active Directory Application - to do this select **Settings** and then **Keys**. This screen displays the Passwords (Client Secrets) and Public Keys (Client Certificates) which are associated with this Azure Active Directory Application.

The Public Key associated with the generated Certificate can be uploaded by selecting **Upload Public Key**, selecting the file which should be uploaded (in the example above, this'd be `service-principal.crt`) - and then hitting **Save**.

Allowing the Service Principal to manage the Subscription

Now that we've created the Application within Azure Active Directory and assigned the certificate we're using for authentication, we can now grant the Application permissions to manage the Subscription. To do this, navigate to the **Subscriptions** blade within the Azure Portal (https://portal.azure.com/#blade/Microsoft_Azure_Billing/SubscriptionsBlade), select the Subscription you wish to use, then click **Access Control (IAM)** and finally **Add role assignment**.

Firstly, specify a Role which grants the appropriate permissions needed for the Service Principal (for example, `Contributor` will grant Read/Write on all resources in the Subscription). More information about the built in roles can be found here (https://azure.microsoft.com/en-gb/documentation/articles/role-based-access-built-in-roles/).

Secondly, search for and select the name of the Application created in Azure Active Directory to assign it this role - then press **Save**.

At this point the newly created Azure Active Directory Application should be associated with the Certificate that we generated earlier (which can be used as a Client Certificate) - and should have permissions to the Azure Subscription.

Configuring the Service Principal in Terraform

As we've obtained the credentials for this Service Principal - it's possible to configure them in a few different ways.

When storing the credentials as Environment Variables, for example:

```
$ export ARM_CLIENT_ID="00000000-0000-0000-0000-000000000000"
$ export ARM_CLIENT_CERTIFICATE_PATH="/path/to/my/client/certificate.pfx"
$ export ARM_CLIENT_CERTIFICATE_PASSWORD="Pa55w0rd123"
$ export ARM_SUBSCRIPTION_ID="00000000-0000-0000-0000-000000000000"
$ export ARM_TENANT_ID="00000000-0000-0000-0000-000000000000"
```

The following Provider block can be specified - where `1.20.0` is the version of the Azure Provider that you'd like to use:

```
provider "azuread" {
  # Whilst version is optional, we /strongly recommend/ using it to pin the version of the Provider being
 used
  version = "=0.1.0"
}
```

More information on the fields supported in the Provider block can be found here (/docs/providers/azuread/index.html#argument-reference).

At this point running either `terraform plan` or `terraform apply` should allow Terraform to run using the Service Principal to authenticate.

It's also possible to configure these variables either in-line or from using variables in Terraform (as the `client_certificate_path` and `client_certificate_password` are in this example), like so:

NOTE: We'd recommend not defining these variables in-line since they could easily be checked into Source Control.

```
variable "client_certificate_path" {}
variable "client_certificate_password" {}

provider "azuread" {
  # Whilst version is optional, we /strongly recommend/ using it to pin the version of the Provider being
 used
  version = "=0.1.0"

  subscription_id             = "00000000-0000-0000-0000-000000000000"
  client_id                   = "00000000-0000-0000-0000-000000000000"
  client_certificate_path     = "${var.client_certificate_path}"
  client_certificate_password = "${var.client_certificate_password}"
  tenant_id                   = "00000000-0000-0000-0000-000000000000"
}
```

More information on the fields supported in the Provider block can be found here (/docs/providers/azuread/index.html#argument-reference).

At this point running either `terraform plan` or `terraform apply` should allow Terraform to run using the Service Principal to authenticate.

Next you may want to follow the Granting a Service Principal permission to manage AAD (/docs/providers/azuread/auth/service_principal_configuration.html) guide to grant the Service Ability permission to create and modify Azure Active Directory objects such as users and groups.

Azure Active Directory Provider: Authenticating using a Service Principal with a Client Secret

Terraform supports a number of different methods for authenticating to Azure:

- Authenticating to Azure using the Azure CLI (/docs/providers/azuread/auth/azure_cli.html)
- Authenticating to Azure using Managed Service Identity (/docs/providers/azuread/auth/managed_service_identity.html)
- Authenticating to Azure using a Service Principal and a Client Certificate (/docs/providers/azuread/auth/service_principal_client_certificate.html)
- Authenticating to Azure using a Service Principal and a Client Secret (which is covered in this guide)

Further steps must be taken to grant a Service Principal permission to manage objects in an Azure Active Directory:

Granting a Service Principal permission to manage AAD (/docs/providers/azuread/auth/service_principal_configuration.html)

We recommend using either a Service Principal or Managed Service Identity when running Terraform non-interactively (such as when running Terraform in a CI server) - and authenticating using the Azure CLI when running Terraform locally.

Beyond authentication and managing Azure AAD resources further steps are required to make so a Service principal can make changes to Azure Active Directory objects such as users and groups. The Granting a Service Principal permission to manage AAD (/docs/providers/azuread/auth/service_principal_configuration.html) guide contains the required steps.

Creating a Service Principal

A Service Principal is an application within Azure Active Directory whose authentication tokens can be used as the `client_id`, `client_secret`, and `tenant_id` fields needed by Terraform (`subscription_id` can be independently recovered from your Azure account details).

It's possible to complete this task in either the Azure CLI or in the Azure Portal - in both we'll create a Service Principal which has `Contributor` rights to the subscription. It's also possible to assign other rights (https://azure.microsoft.com/en-gb/documentation/articles/role-based-access-built-in-roles/) depending on your configuration.

Creating a Service Principal using the Azure CLI

Note: If you're using the **China**, **German** or **Government** Azure Clouds - you'll need to first configure the Azure CLI to work with that Cloud. You can do this by running:

```
$ az cloud set --name AzureChinaCloud|AzureGermanCloud|AzureUSGovernment
```

Firstly, login to the Azure CLI using:

```
$ az login
```

Once logged in - it's possible to list the Subscriptions associated with the account via:

```
$ az account list
```

The output (similar to below) will display one or more Subscriptions - with the `id` field being the `subscription_id` field referenced above.

```
[
  {
    "cloudName": "AzureCloud",
    "id": "00000000-0000-0000-0000-000000000000",
    "isDefault": true,
    "name": "PAYG Subscription",
    "state": "Enabled",
    "tenantId": "00000000-0000-0000-0000-000000000000",
    "user": {
      "name": "user@example.com",
      "type": "user"
    }
  }
]
```

Should you have more than one Subscription, you can specify the Subscription to use via the following command:

```
$ az account set --subscription="SUBSCRIPTION_ID"
```

We can now create the Service Principal which will have permissions to manage resources in the specified Subscription using the following command:

```
$ az ad sp create-for-rbac --role="Contributor" --scopes="/subscriptions/SUBSCRIPTION_ID"
```

This command will output 5 values:

```
{
  "appId": "00000000-0000-0000-0000-000000000000",
  "displayName": "azure-cli-2017-06-05-10-41-15",
  "name": "http://azure-cli-2017-06-05-10-41-15",
  "password": "0000-0000-0000-0000-000000000000",
  "tenant": "00000000-0000-0000-0000-000000000000"
}
```

These values map to the Terraform variables like so:

- `appId` is the `client_id` defined above.
- `password` is the `client_secret` defined above.
- `tenant` is the `tenant_id` defined above.

Finally, it's possible to test these values work as expected by first logging in:

```
$ az login --service-principal -u CLIENT_ID -p CLIENT_SECRET --tenant TENANT_ID
```

Once logged in as the Service Principal - we should be able to list the VM sizes by specifying an Azure region, for example here we use the `West US` region:

```
$ az vm list-sizes --location westus
```

Note: If you're using the **China**, **German** or **Government** Azure Clouds - you will need to switch `westus` out for another region. You can find which regions are available by running:

```
$ az account list-locations
```

Finally, since we're logged into the Azure CLI as a Service Principal we recommend logging out of the Azure CLI (but you can instead log in using your user account):

```
$ az logout
```

Information on how to configure the Provider block using the newly created Service Principal credentials can be found below.

Creating a Service Principal in the Azure Portal

There are three tasks necessary to create a Service Principal using the Azure Portal (https://portal.azure.com):

1. Create an Application in Azure Active Directory (which acts as a Service Principal)
2. Generating a Client Secret for the Azure Active Directory Application (which can be used for authentication)
3. Grant the Application access to manage resources in your Azure Subscription

1. Creating an Application in Azure Active Directory

Firstly navigate to the **Azure Active Directory** overview (https://portal.azure.com/#blade/Microsoft_AAD_IAM/ActiveDirectoryMenuBlade/Overview) within the Azure Portal - then select the **App Registration** blade (https://portal.azure.com/#blade/Microsoft_AAD_IAM/ActiveDirectoryMenuBlade/RegisteredApps/RegisteredApps/Overview) and click **Endpoints** at the top of the **App Registration** blade. A list of URIs will be displayed and you need to locate the URI for **OAUTH 2.0 AUTHORIZATION ENDPOINT** which contains a GUID. This GUID is your Tenant ID (the `tenant_id` field mentioned above).

Next, navigate back to the **App Registration** blade (https://portal.azure.com/#blade/Microsoft_AAD_IAM/ActiveDirectoryMenuBlade/RegisteredApps/RegisteredApps/Overview) - from here we'll create the Application in Azure Active Directory. To do this click **New application registration** at the top to add a new Application within Azure Active Directory. On this page, set the following values then press **Create**:

- **Name** - this is a friendly identifier and can be anything (e.g. "Terraform")
- **Application Type** - this should be set to "Web app / API"
- **Sign-on URL** - this can be anything, providing it's a valid URI (e.g. https://terra.form (https://terra.form))

At this point the newly created Azure Active Directory application should be visible on-screen - if it's not, navigate to the the **App Registration** blade (https://portal.azure.com/#blade/Microsoft_AAD_IAM/ActiveDirectoryMenuBlade/RegisteredApps/RegisteredApps/Overview) and select the Azure Active Directory application. At the top of this page, the "Application ID" GUID is the `client_id` you'll need.

2. Generating a Client Secret for the Azure Active Directory Application

Now that the Azure Active Directory Application exists we can create a Client Secret which can be used for authentication - to do this select **Settings** and then **Keys**. This screen displays the Passwords (Client Secrets) and Public Keys (Client Certificates) which are associated with this Azure Active Directory Application.

On this screen we can generate a new Password by entering a Description and selecting an Expiry Date, and then pressing **Save**. Once the Password has been generated it will be displayed on screen - *the Password is only displayed once* so **be sure to copy it now** (otherwise you will need to regenerate a new key). This newly generated Password is the `client_secret` you will need.

3. Granting the Application access to manage resources in your Azure Subscription

Once the Application exists in Azure Active Directory - we can grant it permissions to modify resources in the Subscription. To do this, navigate to the **Subscriptions** blade within the Azure Portal (https://portal.azure.com/#blade/Microsoft_Azure_Billing/SubscriptionsBlade), then select the Subscription you wish to use, then click **Access Control (IAM)**, and finally **Add role assignment**.

Firstly, specify a Role which grants the appropriate permissions needed for the Service Principal (for example, `Contributor` will grant Read/Write on all resources in the Subscription). There's more information about the built in roles available here (https://azure.microsoft.com/en-gb/documentation/articles/role-based-access-built-in-roles/).

Secondly, search for and select the name of the Application created in Azure Active Directory to assign it this role - then press **Save**.

Configuring the Service Principal in Terraform

As we've obtained the credentials for this Service Principal - it's possible to configure them in a few different ways.

When storing the credentials as Environment Variables, for example:

```
$ export ARM_CLIENT_ID="00000000-0000-0000-0000-000000000000"
$ export ARM_CLIENT_SECRET="00000000-0000-0000-0000-000000000000"
$ export ARM_SUBSCRIPTION_ID="00000000-0000-0000-0000-000000000000"
$ export ARM_TENANT_ID="00000000-0000-0000-0000-000000000000"
```

The following Provider block can be specified - where `1.20.0` is the version of the Azure Provider that you'd like to use:

```
provider "azuread" {
  # Whilst version is optional, we /strongly recommend/ using it to pin the version of the Provider being
 used
  version = "=0.1.0"
}
```

More information on the fields supported in the Provider block can be found here (/docs/providers/azuread/index.html#argument-reference).

At this point running either `terraform plan` or `terraform apply` should allow Terraform to run using the Service Principal to authenticate.

It's also possible to configure these variables either in-line or from using variables in Terraform (as the `client_secret` is in this example), like so:

NOTE: We'd recommend not defining these variables in-line since they could easily be checked into Source Control.

```
variable "client_secret" {}

provider "azuread" {
  # Whilst version is optional, we /strongly recommend/ using it to pin the version of the Provider being
 used
  version = "=0.1.0"

  subscription_id = "00000000-0000-0000-0000-000000000000"
  client_id       = "00000000-0000-0000-0000-000000000000"
  client_secret   = "${var.client_secret}"
  tenant_id       = "00000000-0000-0000-0000-000000000000"
}
```

More information on the fields supported in the Provider block can be found here (/docs/providers/azuread/index.html#argument-reference).

At this point running either `terraform plan` or `terraform apply` should allow Terraform to run using the Service Principal to authenticate.

Next you may want to follow the Granting a Service Principal permission to manage AAD (/docs/providers/azuread/auth/service_principal_configuration.html) guide to grant the Service Ability permission to create and modify Azure Active Directory objects such as users and groups.

Azure Active Directory Provider: Configuring a Service Principal for managing Azure Active Directory

Terraform supports a number of different methods for authenticating to Azure:

- Authenticating to Azure using the Azure CLI (/docs/providers/azuread/auth/azure_cli.html)
- Authenticating to Azure using Managed Service Identity (/docs/providers/azuread/auth/managed_service_identity.html)
- Authenticating to Azure using a Service Principal and a Client Certificate (/docs/providers/azuread/auth/service_principal_client_certificate.html)
- Authenticating to Azure using a Service Principal and a Client Secret (which is covered in this guide)

Further steps must be taken to grant a Service Principal permission to manage objects in an Azure Active Directory:

- Granting a Service Principal permission to manage AAD (which is covered in this guide)

We recommend using either a Service Principal or Managed Service Identity when running Terraform non-interactively (such as when running Terraform in a CI server) - and authenticating using the Azure CLI when running Terraform locally.

Creating a Service Principal

A Service Principal is an application within Azure Active Directory whose authentication tokens can be used as the `client_id`, `client_secret`, and `tenant_id` fields needed by Terraform (`subscription_id` can be independently recovered from your Azure account details).

Depending on how the service principal authenticates to azure it can be created in a number of different ways: * Authenticating to Azure using a Service Principal and a Client Certificate (/docs/providers/azuread/auth/service_principal_client_certificate.html) * Authenticating to Azure using a Service Principal and a Client Secret (/docs/providers/azuread/auth/service_principal_client_secret.html)

Granting administrator permissions

Note: This requires the use of powershell cmdlets and is easiest to run in CloudShell.

Firstly, connect to the directory using:

```
Connect-AzureAD -TenantID "00000000-0000-0000-0000-000000000000"
```

Next we want to get the correct role to assign, in this case `User Account Administrator`:

```
$role = Get-AzureADDirectoryRole | Where-Object {$_.displayName -eq 'User Account Administrator'}
Write-Host $role
```

Since this is a built-in Role, if this doesn't exist (returns `null` above) then we need to instantiate it from the Role Template:

```
if ($role -eq $null) {
    # Instantiate an instance of the role template
    $roleTemplate = Get-AzureADDirectoryRoleTemplate | Where-Object {$_.displayName -eq 'User Account Adm
inistrator'}
    Enable-AzureADDirectoryRole -RoleTemplateId $roleTemplate.ObjectId

    # Fetch User Account Administrator role instance again
    $role = Get-AzureADDirectoryRole | Where-Object {$_.displayName -eq 'User Account Administrator'}
}
```

Next we need the Client ID (sometimes referred to as the Application ID) of the Service Principal. We can look this up by it's display name:

```
$sp = Get-AzureADServicePrincipal -All $true | Where-Object {$_.displayName -eq 'Service Principal Name'}
$sp.ObjectId
```

Now that we have all the required information we can add the service principal to the role:

```
Add-AzureADDirectoryRoleMember -ObjectId $role.ObjectId -RefObjectId $sp.ObjectId
```

Finally we can repeat this for the `Company Administrator` role:

```
$role = Get-AzureADDirectoryRole | Where-Object {$_.displayName -eq 'Company Administrator'}
$role

if ($role -eq $null) {
    # Instantiate an instance of the role template
    $roleTemplate = Get-AzureADDirectoryRoleTemplate | Where-Object {$_.displayName -eq 'Company Administ
rator'}
    Enable-AzureADDirectoryRole -RoleTemplateId $roleTemplate.ObjectId

    # Fetch User Account Administrator role instance again
    $role = Get-AzureADDirectoryRole | Where-Object {$_.displayName -eq 'Company Administrator'}
}

$sp = Get-AzureADServicePrincipal | Where-Object {$_.displayName -eq 'Service Principal Name'}
$sp.ObjectId

Add-AzureADDirectoryRoleMember -ObjectId $role.ObjectId -RefObjectId $sp.ObjectId
```

At this point you should now be able to manage Users, Groups and other Azure Active Directory resources using Terraform.

Data Source: azuread_application

Use this data source to access information about an existing Application within Azure Active Directory.

NOTE: If you're authenticating using a Service Principal then it must have permissions to both `Read and write all (or owned by) applications` and `Sign in and read user profile` within the `Windows Azure Active Directory` API.

Example Usage

```
data "azuread_application" "example" {
  name = "My First AzureAD Application"
}

output "azure_ad_object_id" {
  value = "${data.azuread_application.example.id}"
}
```

Argument Reference

- `object_id` - (Optional) Specifies the Object ID of the Application within Azure Active Directory.
- `name` - (Optional) Specifies the name of the Application within Azure Active Directory.

NOTE: Either an `object_id` or `name` must be specified.

Attributes Reference

- `id` - the Object ID of the Azure Active Directory Application.
- `application_id` - the Application ID of the Azure Active Directory Application.
- `available_to_other_tenants` - Is this Azure AD Application available to other tenants?
- `identifier_uris` - A list of user-defined URI(s) that uniquely identify a Web application within it's Azure AD tenant, or within a verified custom domain if the application is multi-tenant.
- `oauth2_allow_implicit_flow` - Does this Azure AD Application allow OAuth2.0 implicit flow tokens?
- `object_id` - the Object ID of the Azure Active Directory Application.
- `reply_urls` - A list of URLs that user tokens are sent to for sign in, or the redirect URIs that OAuth 2.0 authorization codes and access tokens are sent to.
- `group_membership_claims` - The `groups` claim issued in a user or OAuth 2.0 access token that the app expects.

- `required_resource_access` - A collection of `required_resource_access` blocks as documented below.
- `oauth2_permissions` - A collection of OAuth 2.0 permission scopes that the web API (resource) app exposes to client apps. Each permission is covered by a `oauth2_permission` block as documented below.
- `app_roles` - A collection of `app_role` blocks as documented below. For more information https://docs.microsoft.com/en-us/azure/architecture/multitenant-identity/app-roles (https://docs.microsoft.com/en-us/azure/architecture/multitenant-identity/app-roles)

`required_resource_access` block exports the following:

- `resource_app_id` - The unique identifier for the resource that the application requires access to.
- `resource_access` - A collection of `resource_access` blocks as documented below

`resource_access` block exports the following:

- `id` - The unique identifier for one of the `OAuth2Permission` or `AppRole` instances that the resource application exposes.
- `type` - Specifies whether the id property references an `OAuth2Permission` or an `AppRole` .

`oauth2_permission` block exports the following:

- `id` - The unique identifier for one of the `OAuth2Permission`
- `type` - The type of the permission
- `admin_consent_description` - The description of the admin consent
- `admin_consent_display_name` - The display name of the admin consent
- `is_enabled` - Is this permission enabled?
- `user_consent_description` - The description of the user consent
- `user_consent_display_name` - The display name of the user consent
- `value` - The name of this permission

`app_role` block exports the following:

- `id` - The unique identifier of the `app_role` .
- `allowed_member_types` - Specifies whether this app role definition can be assigned to users and groups, or to other applications (that are accessing this application in daemon service scenarios). Possible values are: `User` and `Application` , or both.
- `description` - Permission help text that appears in the admin app assignment and consent experiences.
- `display_name` - Display name for the permission that appears in the admin consent and app assignment experiences.
- `is_enabled` - Determines if the app role is enabled.

- `value` - Specifies the value of the roles claim that the application should expect in the authentication and access tokens.

Data Source: azuread_domains

Use this data source to access information about an existing Domains within Azure Active Directory.

NOTE: If you're authenticating using a Service Principal then it must have permissions to `Directory.Read.All` within the `Windows Azure Active Directory` API.

Example Usage

```
data "azuread_domains" "aad_domains" {}

output "domains" {
  value = "${data.azuread_domains.aad_domains.domains}"
}
```

Argument Reference

- `include_unverified` - (Optional) Set to `true` if unverified Azure AD Domains should be included. Defaults to `false`.
- `only_default` - (Optional) Set to `true` to only return the default domain.
- `only_initial` - (Optional) Set to `true` to only return the initial domain, which is your primary Azure Active Directory tenant domain. Defaults to `false`.

NOTE: If `include_unverified` is set to `true` you cannot specify `only_default` or `only_initial`. Additionally you cannot combine `only_default` with `only_initial`.

Attributes Reference

- `domains` - One or more `domain` blocks as defined below.

The `domain` block contains:

- `domain_name` - The name of the domain.
- `authentication_type` - The authentication type of the domain (Managed or Federated).
- `is_default` - `True` if this is the default domain that is used for user creation.
- `is_initial` - `True` if this is the initial domain created by Azure Activie Directory.
- `is_verified` - `True` if the domain has completed domain ownership verification.

Data Source: azuread_group

Gets information about an Azure Active Directory group.

NOTE: If you're authenticating using a Service Principal then it must have permissions to `Read directory data` within the `Windows Azure Active Directory` API.

Example Usage (by Group Display Name)

```
data "azuread_group" "example" {
  name = "A-AD-Group"
}
```

Argument Reference

The following arguments are supported:

- `name` - (Optional) The Name of the AD Group we want to lookup.
- `object_id` - (Optional) Specifies the Object ID of the AD Group within Azure Active Directory.

NOTE: Either a `name` or an `object_id` must be specified.

Attributes Reference

The following attributes are exported:

- `id` - The Object ID of the Azure AD Group.

Data Source: azuread_user

Gets Object IDs or Display Names for multiple Azure Active Directory groups.

NOTE: If you're authenticating using a Service Principal then it must have permissions to `Read directory data` within the `Windows Azure Active Directory` API.

Example Usage

```
data "azuread_groups" "groups" {
  names = ["group-a", "group-b"]
}
```

Argument Reference

The following arguments are supported:

- `names` - (optional) The Display Names of the Azure AD Groups.
- `object_ids` - (Optional) The Object IDs of the Azure AD Groups.

NOTE: Either `names` or `object_ids` must be specified.

Attributes Reference

The following attributes are exported:

- `object_ids` - The Object IDs of the Azure AD Groups.
- `names` - The Display Names of the Azure AD Groups.

Data Source: azuread_service_principal

Gets information about an existing Service Principal associated with an Application within Azure Active Directory.

NOTE: If you're authenticating using a Service Principal then it must have permissions to both `Read and write all applications` and `Sign in and read user profile` within the `Windows Azure Active Directory` API.

Example Usage (by Application Display Name)

```
data "azuread_service_principal" "example" {
  display_name = "my-awesome-application"
}
```

Example Usage (by Application ID)

```
data "azuread_service_principal" "example" {
  application_id = "00000000-0000-0000-0000-000000000000"
}
```

Example Usage (by Object ID)

```
data "azuread_service_principal" "example" {
  object_id = "00000000-0000-0000-0000-000000000000"
}
```

Argument Reference

The following arguments are supported:

- `application_id` - (Optional) The ID of the Azure AD Application.
- `object_id` - (Optional) The ID of the Azure AD Service Principal.
- `display_name` - (Optional) The Display Name of the Azure AD Application associated with this Service Principal.

NOTE: At least one of `application_id`, `display_name` or `object_id` must be specified.

- `app_roles` - A collection of `app_role` blocks as documented below. For more information https://docs.microsoft.com/en-us/azure/architecture/multitenant-identity/app-roles (https://docs.microsoft.com/en-us/azure/architecture/multitenant-identity/app-roles)

- `oauth2_permissions` - A collection of OAuth 2.0 permissions exposed by the associated application. Each permission is covered by a `oauth2_permission` block as documented below.

Attributes Reference

The following attributes are exported:

- `id` - The Object ID for the Service Principal.

`oauth2_permission` block exports the following:

- `id` - The unique identifier for one of the `OAuth2Permission`
- `type` - The type of the permission
- `admin_consent_description` - The description of the admin consent
- `admin_consent_display_name` - The display name of the admin consent
- `is_enabled` - Is this permission enabled?
- `user_consent_description` - The description of the user consent
- `user_consent_display_name` - The display name of the user consent
- `value` - The name of this permission

`app_role` block exports the following:

- `id` - The unique identifier of the `app_role`.
- `allowed_member_types` - Specifies whether this app role definition can be assigned to users and groups, or to other applications (that are accessing this application in daemon service scenarios). Possible values are: `User` and `Application`, or both.
- `description` - Permission help text that appears in the admin app assignment and consent experiences.
- `display_name` - Display name for the permission that appears in the admin consent and app assignment experiences.
- `is_enabled` - Determines if the app role is enabled.
- `value` - Specifies the value of the roles claim that the application should expect in the authentication and access tokens.

Data Source: azuread_user

Gets information about an Azure Active Directory user.

NOTE: If you're authenticating using a Service Principal then it must have permissions to `Read directory data` within the `Windows Azure Active Directory` API.

Example Usage

```
data "azuread_user" "example" {
  user_principal_name = "user@hashicorp.com"
}
```

Argument Reference

The following arguments are supported:

- `user_principal_name` - (Required) The User Principal Name of the Azure AD User.
- `object_id` - (Optional) Specifies the Object ID of the Application within Azure Active Directory.

NOTE: Either a `user_principal_name` or an `object_id` must be specified.

Attributes Reference

The following attributes are exported:

- `id` - The Object ID of the Azure AD User.
- `user_principal_name` - The User Principal Name of the Azure AD User.
- `account_enabled` - `True` if the account is enabled; otherwise `False`.
- `display_name` - The Display Name of the Azure AD User.
- `mail` - The primary email address of the Azure AD User.
- `mail_nickname` - The email alias of the Azure AD User.

Data Source: azuread_user

Gets Object IDs or UPNs for multiple Azure Active Directory users.

NOTE: If you're authenticating using a Service Principal then it must have permissions to `Read directory data` within the `Windows Azure Active Directory` API.

Example Usage

```
data "azuread_users" "users" {
  user_principal_names = ["kat@hashicorp.com", "byte@hashicorp.com"]
}
```

Argument Reference

The following arguments are supported:

- `user_principal_names` - (optional) The User Principal Names of the Azure AD Users.
- `object_ids` - (Optional) The Object IDs of the Azure AD Users.

NOTE: Either `user_principal_names` or `object_ids` must be specified.

Attributes Reference

The following attributes are exported:

- `object_ids` - The Object IDs of the Azure AD Users.
- `user_principal_names` - The User Principal Names of the Azure AD Users.

azuread_application

Manages an Application within Azure Active Directory.

NOTE: If you're authenticating using a Service Principal then it must have permissions to both `Read and write owned by applications` and `Sign in and read user profile` within the `Windows Azure Active Directory` API.

Example Usage

```
resource "azuread_application" "example" {
  name                       = "example"
  homepage                   = "https://homepage"
  identifier_uris            = ["https://uri"]
  reply_urls                 = ["https://replyurl"]
  available_to_other_tenants = false
  oauth2_allow_implicit_flow = true
  type                       = "webapp/api"

  required_resource_access {
    resource_app_id = "00000003-0000-0000-c000-000000000000"

    resource_access {
      id   = "..."
      type = "Role"
    }

    resource_access {
      id   = "..."
      type = "Scope"
    }

    resource_access {
      id   = "..."
      type = "Scope"
    }
  }

  required_resource_access {
    resource_app_id = "00000002-0000-0000-c000-000000000000"

    resource_access {
      id   = "..."
      type = "Scope"
    }
  }

  app_role {
    allowed_member_types = [
      "User",
      "Application",
    ]

    description  = "Admins can manage roles and perform all task actions"
    display_name = "Admin"
    is_enabled   = true
    value        = "Admin"
  }
}
```

Argument Reference

The following arguments are supported:

- `name` - (Required) The display name for the application.

- `homepage` - (optional) The URL to the application's home page. If no homepage is specified this defaults to `https://{name}`.
- `identifier_uris` - (Optional) A list of user-defined URI(s) that uniquely identify a Web application within it's Azure AD tenant, or within a verified custom domain if the application is multi-tenant.
- `reply_urls` - (Optional) A list of URLs that user tokens are sent to for sign in, or the redirect URIs that OAuth 2.0 authorization codes and access tokens are sent to.
- `available_to_other_tenants` - (Optional) Is this Azure AD Application available to other tenants? Defaults to `false`.
- `public_client` - (Optional) Is this Azure AD Application a public client? Defaults to `false`.
- `oauth2_allow_implicit_flow` - (Optional) Does this Azure AD Application allow OAuth2.0 implicit flow tokens? Defaults to `false`.
- `group_membership_claims` - (Optional) Configures the `groups` claim issued in a user or OAuth 2.0 access token that the app expects. Defaults to `SecurityGroup`. Possible values are `None`, `SecurityGroup` or `All`.
- `required_resource_access` - (Optional) A collection of `required_resource_access` blocks as documented below.
- `type` - (Optional) Type of an application: `webapp/api` or `native`. Defaults to `webapp/api`. For `native` apps type `identifier_uris` property can not not be set.
- `app_role` - (Optional) A collection of `app_role` blocks as documented below. For more information https://docs.microsoft.com/en-us/azure/architecture/multitenant-identity/app-roles (https://docs.microsoft.com/en-us/azure/architecture/multitenant-identity/app-roles)

`required_resource_access` supports the following:

- `resource_app_id` - (Required) The unique identifier for the resource that the application requires access to. This should be equal to the appId declared on the target resource application.
- `resource_access` - (Required) A collection of `resource_access` blocks as documented below.

`resource_access` supports the following:

- `id` - (Required) The unique identifier for one of the `OAuth2Permission` or `AppRole` instances that the resource application exposes.
- `type` - (Required) Specifies whether the id property references an `OAuth2Permission` or an `AppRole`. Possible values are `Scope` or `Role`.

`app_role` supports the following:

- `id` - The unique identifier of the `app_role`.
- `allowed_member_types` - (Required) Specifies whether this app role definition can be assigned to users and groups by setting to `User`, or to other applications (that are accessing this application in daemon service scenarios) by setting to `Application`, or to both.
- `description` - (Required) Permission help text that appears in the admin app assignment and consent experiences.

- `display_name` - (Required) Display name for the permission that appears in the admin consent and app assignment experiences.
- `is_enabled` - (Optional) Determines if the app role is enabled: Defaults to `true`.
- `value` - (Required) Specifies the value of the roles claim that the application should expect in the authentication and access tokens.

Attributes Reference

The following attributes are exported:

- `application_id` - The Application ID.
- `object_id` - The Application's Object ID.
- `oauth2_permissions` - A collection of OAuth 2.0 permission scopes that the web API (resource) app exposes to client apps. Each permission is covered by a `oauth2_permission` block as documented below.

`oauth2_permission` block exports the following:

- `id` - The unique identifier for one of the `OAuth2Permission`.
- `type` - The type of the permission.
- `admin_consent_description` - The description of the admin consent.
- `admin_consent_display_name` - The display name of the admin consent.
- `is_enabled` - Is this permission enabled?
- `user_consent_description` - The description of the user consent.
- `user_consent_display_name` - The display name of the user consent.
- `value` - The name of this permission.

Import

Azure Active Directory Applications can be imported using the `object id`, e.g.

```
terraform import azuread_application.test 00000000-0000-0000-0000-000000000000
```

azuread_application_password

Manages a Password associated with an Application within Azure Active Directory.

NOTE: If you're authenticating using a Service Principal then it must have permissions to both `Read and write all applications` and `Sign in and read user profile` within the `Windows Azure Active Directory` API.

Example Usage

```
resource "azuread_application" "example" {
  name                       = "example"
  homepage                   = "http://homepage"
  identifier_uris            = ["http://uri"]
  reply_urls                 = ["http://replyurl"]
  available_to_other_tenants = false
  oauth2_allow_implicit_flow = true
}

resource "azuread_application_password" "example" {
  application_id = "${azuread_application.example.id}"
  value          = "VT=uSgbTanZhyz@%nL9Hpd+Tfay_MRV#"
  end_date       = "2020-01-01T01:02:03Z"
}
```

Argument Reference

The following arguments are supported:

- `application_object_id` - (Required) The Object ID of the Application for which this password should be created. Changing this field forces a new resource to be created.
- `value` - (Required) The Password for this Application .
- `end_date` - (Optional) The End Date which the Password is valid until, formatted as a RFC3339 date string (e.g. `2018-01-01T01:02:03Z`). Changing this field forces a new resource to be created.
- `end_date_relative` - (Optional) A relative duration for which the Password is valid until, for example `240h` (10 days) or `2400h30m` . Changing this field forces a new resource to be created.

NOTE: One of `end_date` or `end_date_relative` must be set.

- `key_id` - (Optional) A GUID used to uniquely identify this Password. If not specified a GUID will be created. Changing this field forces a new resource to be created.
- `start_date` - (Optional) The Start Date which the Password is valid from, formatted as a RFC3339 date string (e.g. `2018-01-01T01:02:03Z`). If this isn't specified, the current date is used. Changing this field forces a new resource to be created.

Attributes Reference

The following attributes are exported:

- `id` - The Key ID for the Password.

Import

Passwords can be imported using the `object id` of an Application, e.g.

```
terraform import azuread_application_password.test 00000000-0000-0000-0000-000000000000/11111111-1111-1111-1111-111111111111
```

NOTE: This ID format is unique to Terraform and is composed of the Application's Object ID and the Password's Key ID in the format `{ObjectId}/{PasswordKeyId}`.

azuread_group

Manages a Group within Azure Active Directory.

NOTE: If you're authenticating using a Service Principal then it must have permissions to `Read and write all groups` within the `Windows Azure Active Directory` API. In addition it must also have either the `Company Administrator` or `User Account Administrator` Azure Active Directory roles assigned in order to be able to delete groups. You can assign one of the required Azure Active Directory Roles with the **AzureAD PowerShell Module**, which is available for Windows PowerShell or in the Azure Cloud Shell. Please refer to this documentation (https://docs.microsoft.com/en-us/powershell/module/azuread/add-azureaddirectoryrolemember) for more details.

Example Usage

Basic example

```
resource "azuread_group" "example" {
  name = "A-AD-Group"
}
```

A group with members

```
resource "azuread_user" "example" {
  display_name          = "J Doe"
  password              = "notSecure123"
  user_principal_name   = "j.doe@terraform.onmicrosoft.com"
}

resource "azuread_group" "example" {
  name    = "MyGroup"
  members = [ "${azuread_user.example.object_id}" /*, more users */ ]
}
```

Argument Reference

The following arguments are supported:

- `name` - (Required) The display name for the Group. Changing this forces a new resource to be created.
- `members` (Optional) A set of members who should be present in this Group. Supported Object types are Users, Groups or Service Principals.
- `owners` (Optional) A set of owners who own this Group. Supported Object types are Users or Service Principals.

NOTE: Group names are not unique within Azure Active Directory.

NOTE: Do not use `azuread_group_member` at the same time as the `members` argument.

NOTE: Do not use `azuread_group_owner` at the same time as the `owners` argument.

Attributes Reference

The following attributes are exported:

- `id` - The Object ID of the Group.
- `name` - The Display Name of the Group.
- `members` - The Members of the Group.
- `owners` - The Members of the Group.

Import

Azure Active Directory Groups can be imported using the `object id`, e.g.

```
terraform import azuread_group.my_group 00000000-0000-0000-0000-000000000000
```

azuread_group_member

Manages a single Group Membership within Azure Active Directory.

NOTE: Do not use this resource at the same time as `azuread_group.members`.

Example Usage

```
data "azuread_user" "example" {
  user_principal_name = "jdoe@hashicorp.com"
}

resource "azuread_group" "example" {
  name = "my_group"
}

resource "azuread_group_member" "example" {
  group_object_id   = "${azuread_group.example.id}"
  member_object_id  = "${data.azuread_user.example.id}"
}
```

Argument Reference

The following arguments are supported:

- `group_object_id` - (Required) The Object ID of the Azure AD Group you want to add the Member to. Changing this forces a new resource to be created.
- `member_object_id` - (Required) The Object ID of the Azure AD Object you want to add as a Member to the Group. Supported Object types are Users, Groups or Service Principals. Changing this forces a new resource to be created.

NOTE: The Member object has to be present in your Azure Active Directory, either as a Member or a Guest.

Attributes Reference

The following attributes are exported:

- `id` - The ID of the Azure AD Group Member.

Import

Azure Active Directory Group Members can be imported using the `object id`, e.g.

```
terraform import azuread_group_member.test 00000000-0000-0000-0000-000000000000/11111111-1111-1111-1111-1
11111111111
```

NOTE: This ID format is unique to Terraform and is composed of the Azure AD Group Object ID and the target Member Object ID in the format `{GroupObjectID}/{MemberObjectID}`.

azuread_service_principal

Manages a Service Principal associated with an Application within Azure Active Directory.

NOTE: If you're authenticating using a Service Principal then it must have permissions to both `Read and write all applications` and `Sign in and read user profile` within the `Windows Azure Active Directory` API. Please see The Granting a Service Principal permission to manage AAD (/docs/providers/azuread/auth/service_principal_configuration.html) for the required steps.

Example Usage

```
resource "azuread_application" "example" {
  name                       = "example"
  homepage                   = "http://homepage"
  identifier_uris            = ["http://uri"]
  reply_urls                 = ["http://replyurl"]
  available_to_other_tenants = false
  oauth2_allow_implicit_flow = true
}

resource "azuread_service_principal" "example" {
  application_id               = "${azuread_application.example.application_id}"
  app_role_assignment_required  = false

  tags = ["example", "tags", "here"]
}
```

Argument Reference

The following arguments are supported:

- `application_id` - (Required) The ID of the Azure AD Application for which to create a Service Principal.
- `app_role_assignment_required` - (Optional) Does this Service Principal require an AppRoleAssignment to a user or group before Azure AD will issue a user or access token to the application? Defaults to `false`.
- `tags` - (Optional) A list of tags to apply to the Service Principal.

Attributes Reference

The following attributes are exported:

- `id` - The Object ID (internal ID) for the Service Principal.
- `application_id` - The Application ID (appId) for the Service Principal.
- `object_id` - The Service Principal's Object ID.

- `display_name` - The Display Name of the Azure Active Directory Application associated with this Service Principal.
- `app_role_assignment_required` - Whether this Service Principal requires an AppRoleAssignment to a user or group before Azure AD will issue a user or access token to the application.
- `oauth2_permissions` - A collection of OAuth 2.0 permissions exposed by the associated application. Each permission is covered by a `oauth2_permission` block as documented below.

`oauth2_permission` block exports the following:

- `id` - The unique identifier for one of the `OAuth2Permission`.
- `type` - The type of the permission.
- `admin_consent_description` - The description of the admin consent.
- `admin_consent_display_name` - The display name of the admin consent.
- `is_enabled` - Is this permission enabled?
- `user_consent_description` - The description of the user consent.
- `user_consent_display_name` - The display name of the user consent.
- `value` - The name of this permission.

Import

Azure Active Directory Service Principals can be imported using the `object id`, e.g.

```
terraform import azuread_service_principal.test 00000000-0000-0000-0000-000000000000
```

azuread_service_principal_password

Manages a Password associated with a Service Principal within Azure Active Directory.

NOTE: If you're authenticating using a Service Principal then it must have permissions to both `Read and write all applications` and `Sign in and read user profile` within the `Windows Azure Active Directory` API.

Example Usage

```
resource "azuread_application" "example" {
  name                       = "example"
  homepage                   = "http://homepage"
  identifier_uris            = ["http://uri"]
  reply_urls                 = ["http://replyurl"]
  available_to_other_tenants = false
  oauth2_allow_implicit_flow = true
}

resource "azuread_service_principal" "example" {
  application_id = "${azuread_application.example.application_id}"
}

resource "azuread_service_principal_password" "example" {
  service_principal_id = "${azuread_service_principal.test.id}"
  value                = "VT=uSgbTanZhyz@%nL9Hpd+Tfay_MRV#"
  end_date             = "2020-01-01T01:02:03Z"
}
```

Argument Reference

The following arguments are supported:

- `service_principal_id` - (Required) The ID of the Service Principal for which this password should be created. Changing this field forces a new resource to be created.
- `value` - (Required) The Password for this Service Principal.
- `end_date` - (Optional) The End Date which the Password is valid until, formatted as a RFC3339 date string (e.g. `2018-01-01T01:02:03Z`). Changing this field forces a new resource to be created.
- `end_date_relative` - (Optional) A relative duration for which the Password is valid until, for example `240h` (10 days) or `2400h30m`. Changing this field forces a new resource to be created.

NOTE: One of `end_date` or `end_date_relative` must be set.

- `key_id` - (Optional) A GUID used to uniquely identify this Key. If not specified a GUID will be created. Changing this field forces a new resource to be created.

- `start_date` - (Optional) The Start Date which the Password is valid from, formatted as a RFC3339 date string (e.g. `2018-01-01T01:02:03Z`). If this isn't specified, the current date is used. Changing this field forces a new resource to be created.

Attributes Reference

The following attributes are exported:

- `id` - The Key ID for the Service Principal Password.

Import

Service Principal Passwords can be imported using the `object id`, e.g.

```
terraform import azuread_service_principal_password.test 00000000-0000-0000-0000-000000000000/11111111-1111-1111-1111-111111111111
```

NOTE: This ID format is unique to Terraform and is composed of the Service Principal's Object ID and the Service Principal Password's Key ID in the format `{ServicePrincipalObjectId}/{ServicePrincipalPasswordKeyId}`.

azuread_user

Manages a User within Azure Active Directory.

NOTE: If you're authenticating using a Service Principal then it must have permissions to `Directory.ReadWrite.All` within the `Windows Azure Active Directory` API.

Example Usage

```
resource "azuread_user" "example" {
  user_principal_name = "jdo@hashicorp.com"
  display_name        = "J. Doe"
  mail_nickname       = "jdoe"
  password            = "SecretP@sswd99!"
}
```

Argument Reference

The following arguments are supported:

- `user_principal_name` - (Required) The User Principal Name of the Azure AD User.
- `display_name` - (Required) The name to display in the address book for the user.
- `account_enabled` - (Optional) `true` if the account should be enabled, otherwise `false`. Defaults to `true`.
- `mail_nickname` - (Optional) The mail alias for the user. Defaults to the user name part of the User Principal Name.
- `password` - (Required) The password for the User. The password must satisfy minimum requirements as specified by the password policy. The maximum length is 256 characters.
- `force_password_change` - (Optional) `true` if the User is forced to change the password during the next sign-in. Defaults to `false`.

Attributes Reference

The following attributes are exported:

- `object_id` - The Object ID of the Azure AD User.
- `id` - The Object ID of the Azure AD User.
- `mail` - The primary email address of the Azure AD User.

Import

Azure Active Directory Users can be imported using the `object id`, e.g.

```
terraform import azuread_user.my_user 00000000-0000-0000-0000-000000000000
```