Brent Robinson

Follow 114 Followers About

Automating certificate management with Azure and Let's Encrypt

Brent Robinson Apr 21, 2019 · 14 min read

You've received an email reminding you that an SSL/TLS certificate is about to expire, or worse; you discover the certificate has already expired. First, you remember how this happened last year. Second, you desperately search to find which certificate authority you used for this certificate, and where you stored the username and password. Next, you seek authorisation for the expense, raising a purchase order and obtaining approvals. Finally, you begin the arduous task of updating all the systems using this certificate, without the process document you said you'd write last year.

Did that strike a nerve? Or sound like something you wish to avoid?

In this article, we'll explore how to automate SSL/TLS certificate issuance on Microsoft Azure with Let's Encrypt. Let's Encrypt are a certificate authority with a mission to enable ubiquitous usage of HTTPS across the internet by providing free SSL/TLS certificates.

Automated Certificate Management Environment

The Automated Certificate Management Environment (ACME) protocol is designed to automate the certificate issuance. The cost of operations with ACME is so small, certificate authorities such as Let's Encrypt offer genuine, trusted certificates for free. Considering certificate authorities usually charge hundreds of dollars per certificate, the effort to utilise ACME becomes very cost effective. However, as ACME is entirely automated, it's unable to offer Extended Validation (EV) certificates, but for most situations, standard Domain Validation (DV) certificates are well suited.

operate as an ACME client. Let's Encrypt publish a list compatible of tools and libraries. Certbot is an ACME client recommended by Let's Encrypt, which is designed to automate the end-to-end process, from requesting a certificate, to installing it on an application server.

In many scenarios, Certbot is a seamless solution to implement HTTPS on a website quickly. Certbot is perfect for Linux based single-server scenarios. Certbot runs on Linux and configures a certificate on a single server, so certificates must be copied to other servers outside of Certbot if a collection of servers are supporting a single website. After growing beyond a single server or application requiring the certificate, it's worth abstracting the issuance from the configuration process. Have one process for ordering and renewing certificates, and another process for deploying them.

The ACME protocol is formalised by the Internet Engineering Task Force (IETF) under RFC8555. As part of certificate issuance, the client must prove to the certificate authority that it has control of the domain in which it is requesting a certificate. This proof is obtained by either specifying a DNS TXT record for the domain or responding to an HTTP challenge at a specific URL of the website.

ACME on Azure with Azure DevOps

We'll explore how we can use Azure and Azure DevOps together to automate the certificate issuance and configuration processes. We'll use Posh-ACME as our PowerShell-based ACME client, Let's Encrypt as our certificate authority, and we'll complete DNS challenges to prove control of our domain.

There are some prerequisites you'll need:

· An Azure subscription with permission to create new resources.

· An Azure DevOps project which you can create a code repository, build pipeline, and service connection.

· Permission in Azure Active Directory (AAD) to create an application, service principal, and credential.

· Basic knowledge of Azure, Azure DevOps, and PowerShell.

· Basic knowledge of DNS including CNAME and TXT records.

· Ideally, using Azure DNS to host the DNS zone for your website. At the end of this article, we look at how you can have your DNS zone outside of Azure, but in any case, you'll need permission to add and remove DNS records in the zone.

This solution isn't free; but should cost less than AUD$2 per month. If you're new to Azure, the sign-up credit should cover all the resources we'll be creating. Blob storage and key vault costs less than 10 cents per month. DNS varies depending on the traffic to your website, but for over a million DNS queries per month, you'll spend less than AUD$1.50. Automated certificate management for AUD$2 per month — an excellent investment.

Solution overview

Our objective is to use an Azure DevOps pipeline to automate the issuance and renewal of certificates and upload those certificates (and private keys) to Azure Key Vault. Once we have the certificate and key in Azure Key Vault, we can configure them on the application servers.

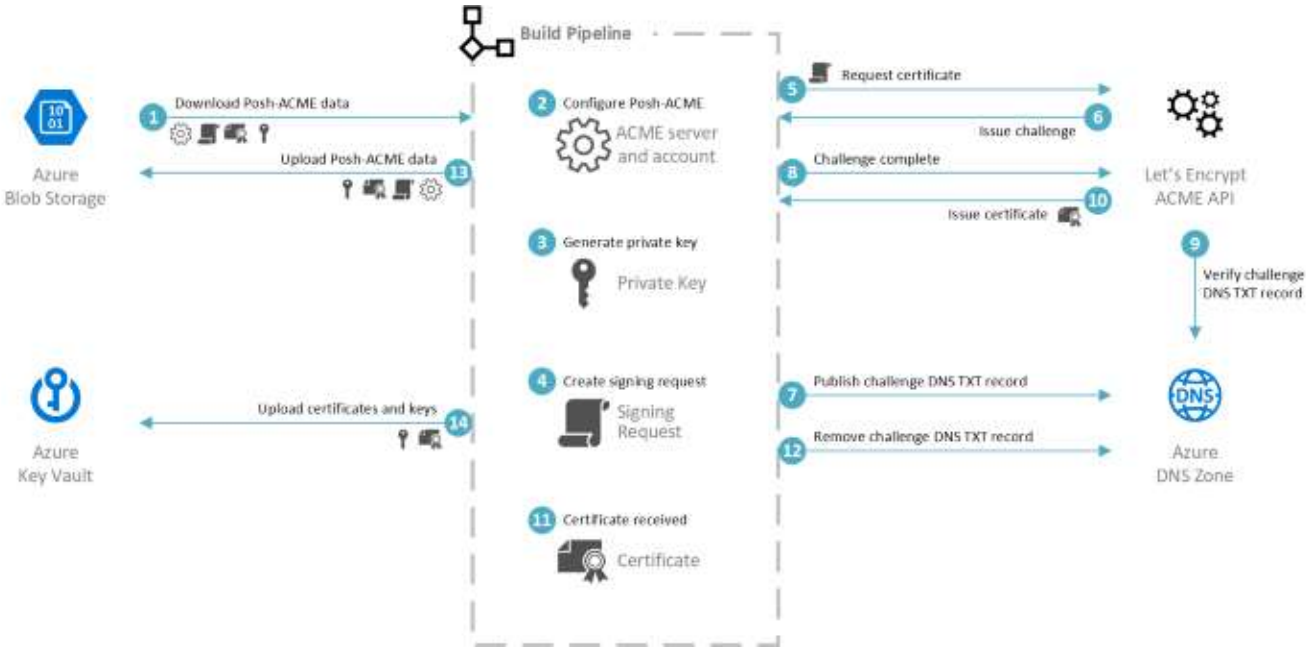


Figure 1: The build pipeline and ACME process for acquiring a certificate

Posh-ACME is designed to orchestrate the issuance with an ACME compatible certificate authority (in our case, Let's Encrypt). Our build pipeline wraps the Posh-

Azure Resources

Let's begin.

We'll need a few resources in Azure to support this process and will create them using Az PowerShell. You could also use the Azure CLI.

Figure 2: The Azure resources required

Start with a resource group if you're not reusing an existing one.

```
New-AzResourceGroup -Name "acme" -Location "australiaeast"
```

DNS Zone: This is where we'll publish our DNS challenges. You'll need to configure your domain registrar to use the DNS zone you've created in Azure. If you're not managing your websites DNS with Azure, there are a few alternatives — check the end of this article for more details. Change the **Name** parameter to your domain name.

```
New-AzDnsZone -Name "brentrobinson.info" -ResourceGroupName "acme"
```

Blob Storage: Posh-ACME is stateful. Posh-ACME stores ACME server information, certificate order details, certificates, and private keys on the file system. We'll persist all this data to Blob storage. We enable *HTTPS Traffic Only* as this storage account contains sensitive private keys which should not be transferred without encryption.

```
New-AzStorageAccount -Name "acmecerts" -ResourceGroupName "acme" -
Location "australiaeast" -Kind StorageV2 -SkuName Standard_LRS -
EnableHttpsTrafficOnly $true

$storageAccountKey = Get-AzStorageAccountKey -Name "acmecerts" -
ResourceGroupName "acme" | Select-Object -First 1 -ExpandProperty
Value

$storageContext = New-AzStorageContext -StorageAccountName
"acmecerts" -StorageAccountKey $storageAccountKey

New-AzStorageContainer -Name "poshacme" -Context $storageContext
```

Key Vault: We'll use Key Vault to store the issued certificates and their private keys. You could alternatively access this information from the Posh-ACME state in blob storage. On the key vault, we enable *purge protection* and *soft delete* to provide additional protection against accidental loss. Change the key vault **Name** parameter to be globally unique.

Note: if you are following along as a learning exercise only, don't enable soft delete or purge protection — skipping these features simplifies resource clean up.

```
New-AzKeyVault -Name "acmecerts" -ResourceGroupName "acme" -Location
"australiaeast" -EnablePurgeProtection -EnableSoftDelete
```

Azure DevOps

We'll use Azure DevOps to orchestrate the certificate issuance process. The build pipeline is scheduled to execute at regular intervals to renew the certificate if required. The build pipeline uses a service connection to control Azure resources. We'll define the build pipeline in YAML format, and store it inside an Azure Repos Git repository.

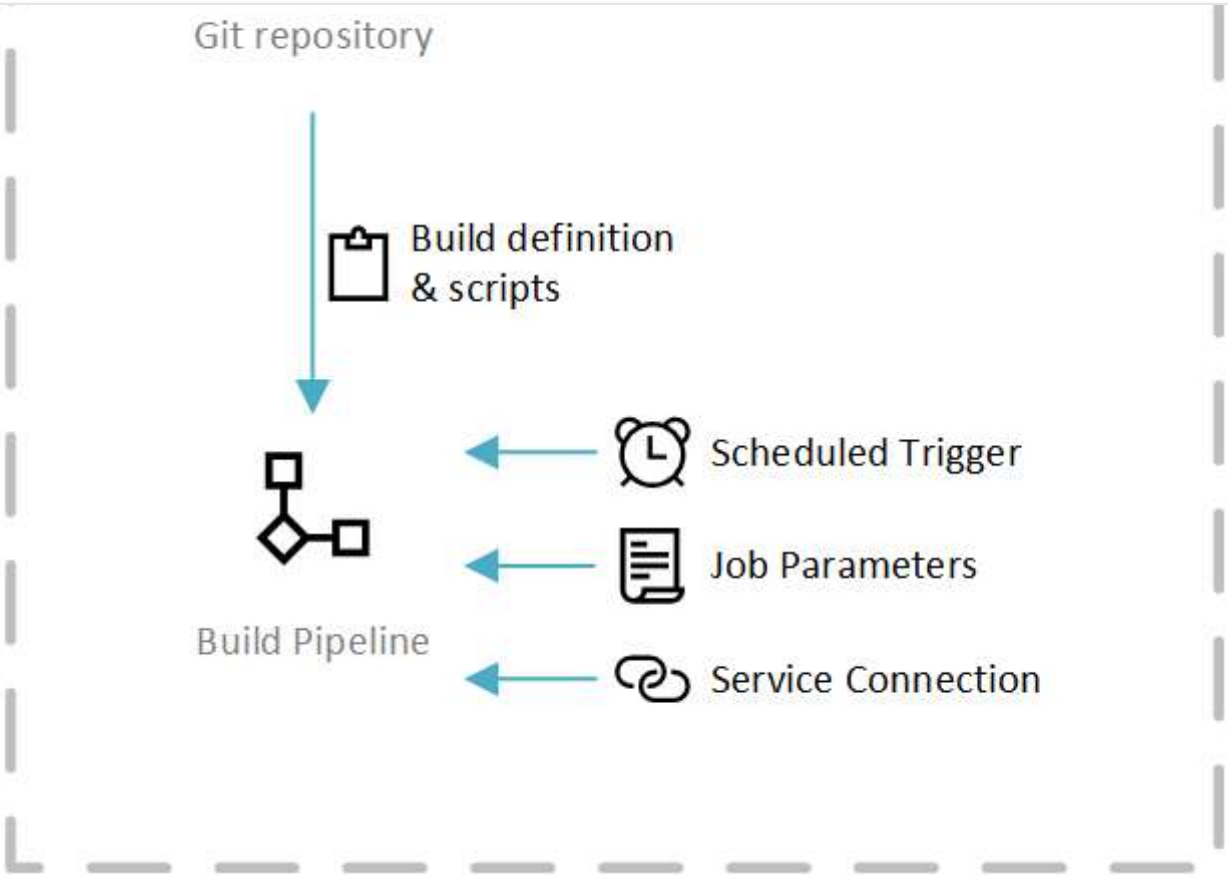


Figure 3: The inputs to the Azure DevOps build pipeline

Create the service principal

We'll create an Azure service principal with just enough permissions to our Azure resources. Again, we'll use Az PowerShell.

```
$application = New-AzADApplication -DisplayName "ACME Certificate
Automation" -IdentifierUris "http://brentrobinson.info/acme"

$servicePrincipal = New-AzADServicePrincipal -ApplicationId
$application.ApplicationId

$servicePrincipalCredential = New-AzADServicePrincipalCredential -
ServicePrincipalObject $servicePrincipal -EndDate (Get-
Date).AddYears(5)
```

The **IdentifierUris** parameter is irrelevant and unused in our scenario, but it is a required parameter, so ensure it's something unique within your Azure Active Directory tenant.

```
Write-Output ("Client ID     = {0}" -f
$servicePrincipal.ApplicationId)

Write-Output ("Client Secret = {0}" -f
[System.Net.NetworkCredential]::new([string]::Empty,
$servicePrincipalCredential.Secret).Password)
```

Record both these values as we'll use them shortly.

Grant the service principal access to the resources

The service principal needs access to the resources created in Azure.

We'll grant **DNS Zone Contributor** on the **DNS Zone** to enable Posh-ACME to create the DNS challenge TXT records for domain validation.

```
New-AzRoleAssignment -ObjectId $servicePrincipal.Id -
ResourceGroupName "acme" -ResourceName "brentrobinson.info" -
ResourceType "Microsoft.Network/dnszones" -RoleDefinitionName "DNS
Zone Contributor"
```

We'll grant the service principal access to read the key vault, on the control plane. We'll also grant access to the data plane. **Get Certificate** allows us to check if we've been issued a newer certificate by examining the thumbprint of the certificate in Key Vault. **Import Certificate** allows us to import issues certificates into the Key Vault.

```
New-AzRoleAssignment -ObjectId $servicePrincipal.Id -
ResourceGroupName "acme" -ResourceName "acmecerts" -ResourceType
"Microsoft.KeyVault/vaults" -RoleDefinitionName "Reader"

Set-AzKeyVaultAccessPolicy -ResourceGroupName "acme" -VaultName
"acmecerts" -ObjectId $servicePrincipal.Id -
PermissionsToCertificates Get, Import
```

We'll generate a SAS token to access the data in the Azure Storage container.

```
value

$storageContext = New-AzStorageContext -StorageAccountName
"acmecerts" -StorageAccountKey $storageAccountKey

New-AzStorageContainerSASToken -Name "poshacme" -Permission rwdl -
Context $storageContext -FullUri -ExpiryTime (Get-Date).AddYears(5)
```

Record the SAS token URI as we'll use it as we configure the build pipeline shortly.

We'll also need to know the resource ID of the key vault, so let's retrieve it now.

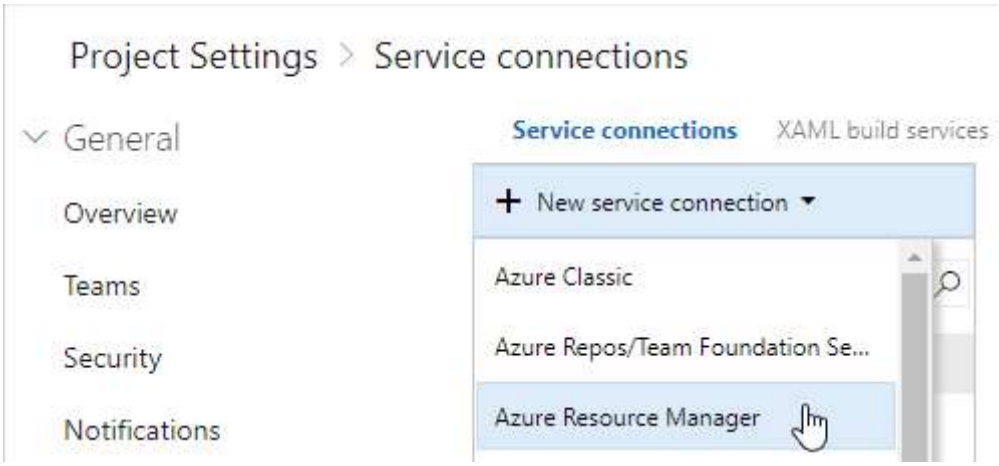
```
Get-AzKeyVault -ResourceGroupName "acme" -VaultName "acmecerts" |
Select-Object -ExpandProperty ResourceId
```

Record the key vault resource ID.

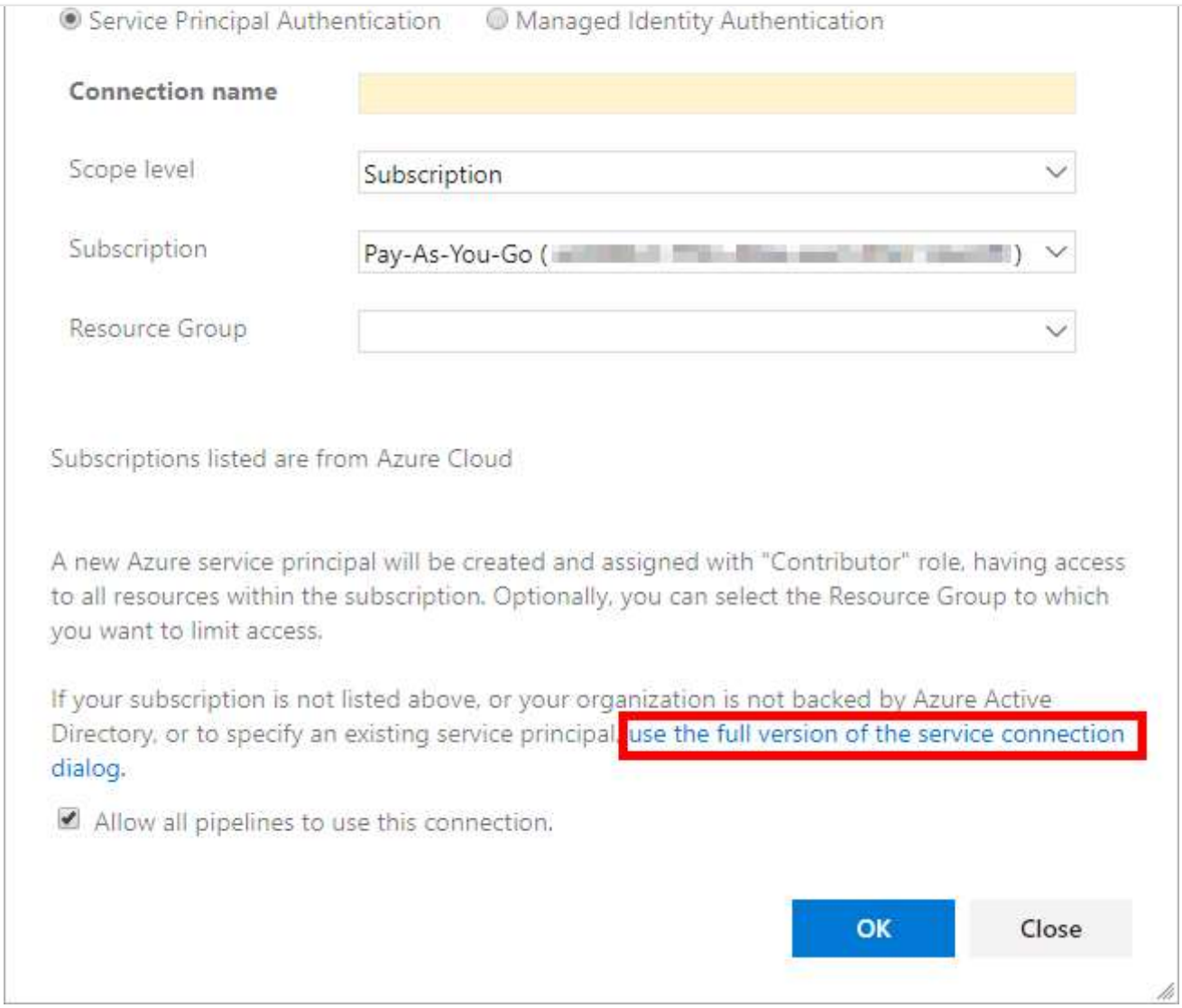
Create a service connection

Now, we'll connect Azure DevOps to Azure using the service principal we've created.

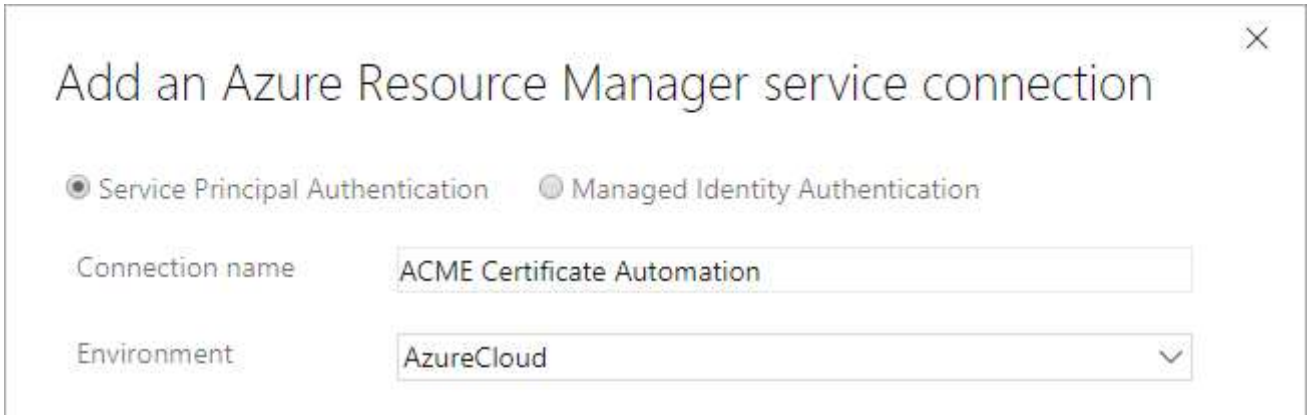
In your Azure DevOps project, navigate to **Project Settings**, and select **Service connections**. From the **New service connection** menu select **Azure Resource Manager**.

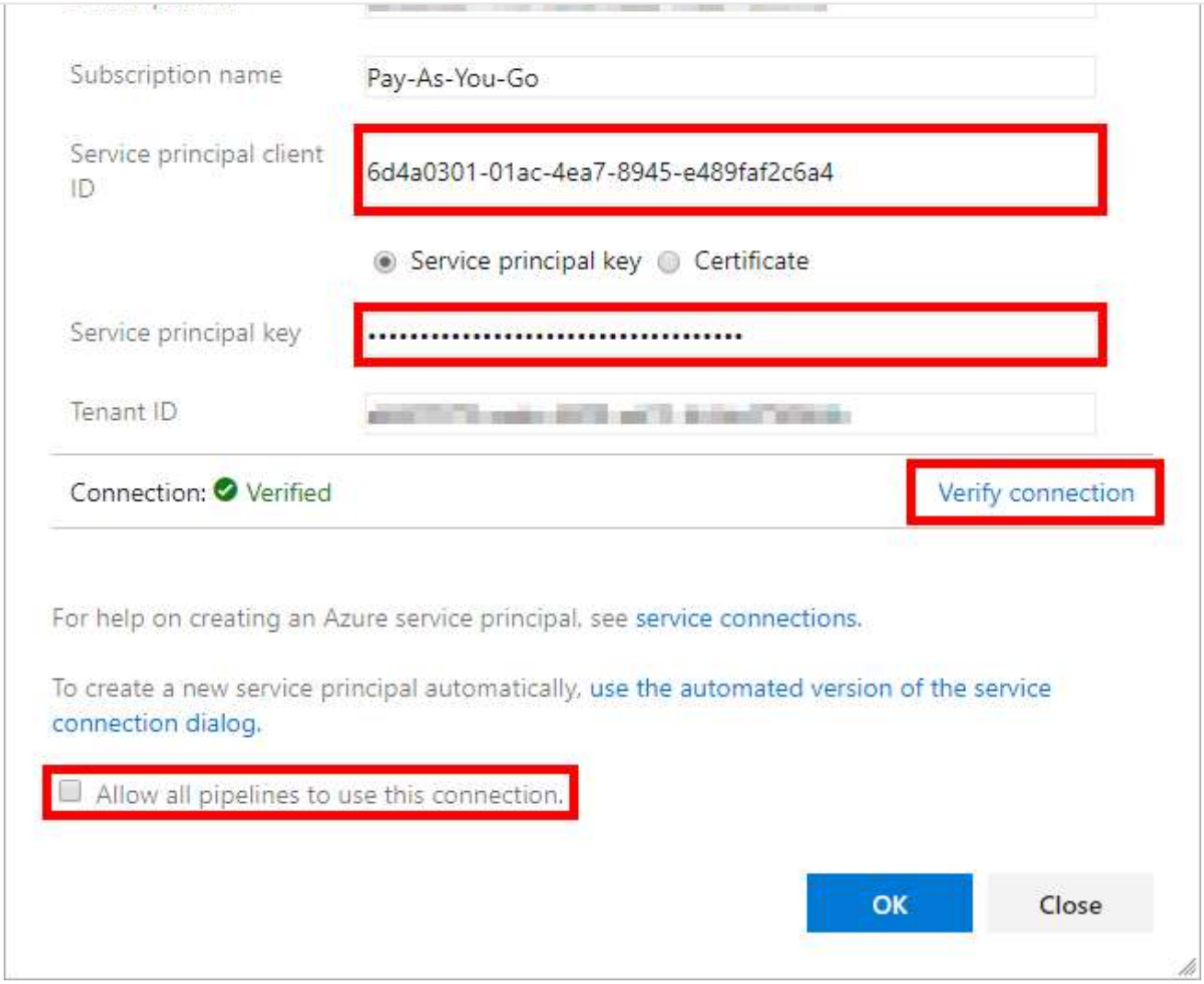


Wait a moment until the **Subscription** field is loaded, select the correct subscription, then click **use the full version of the service connection dialog**.



Enter the **service principal client ID** and **service principal key** into the dialog. The principal key is the password credential we generated when creating the service principal. Click **Verify connection** to validate the details are correct. For security best practice, untick **Allow all pipelines to use this connection**. We'll authorise only the pipeline that needs this connection, which prevents other pipelines from inadvertently gaining access to our certificate and private key data.





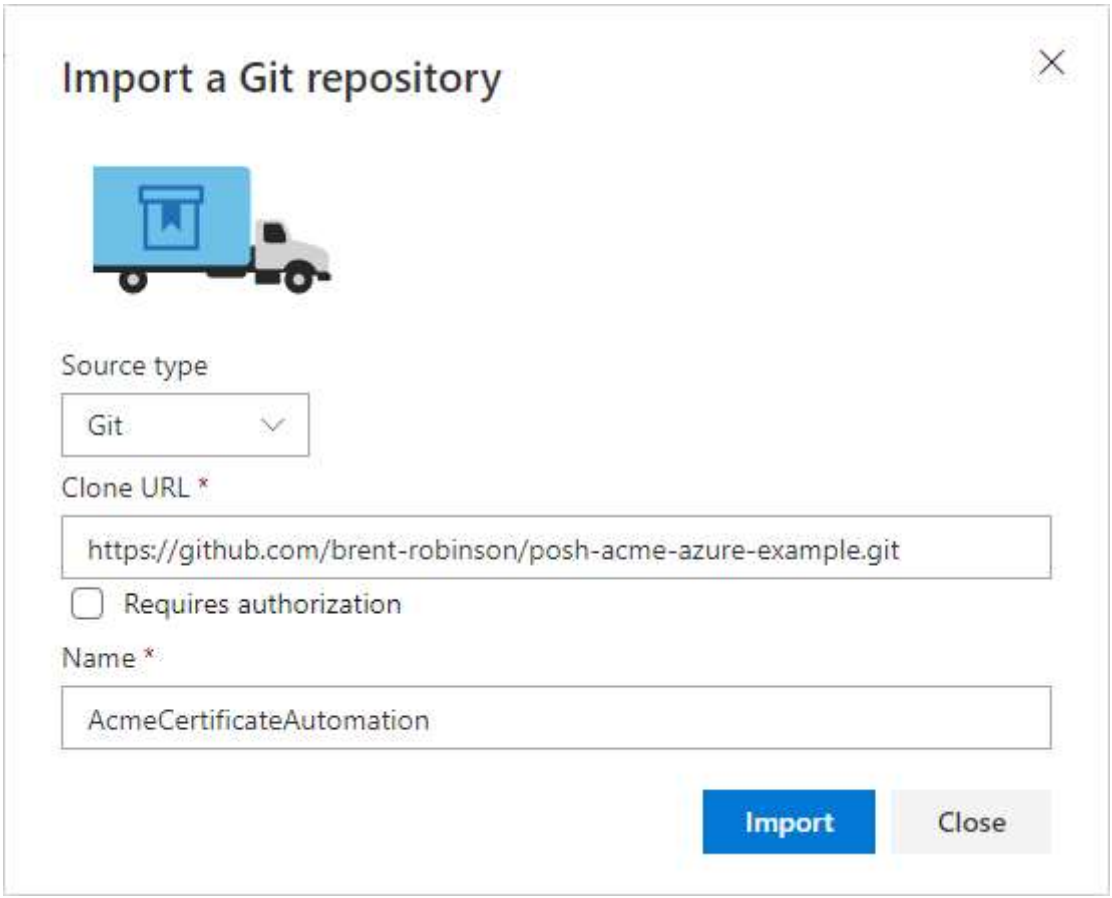
Clone the repository with the pipeline definition

We'll initialise our Azure Repos git repository by cloning an existing repository. While it is possible to reference the existing repository directly from your build pipeline, it's best practice to clone it. Cloning the repository ensures you maintain full control of changes of the scripts; remember these scripts execute on your build agents with your credentials.

In your Azure DevOps project, open **Repos** and select **Import Repository**.

Enter the **Clone URL:** https://github.com/brent-robinson/posh-acme-azure-example.git

Choose a name for your new repository.

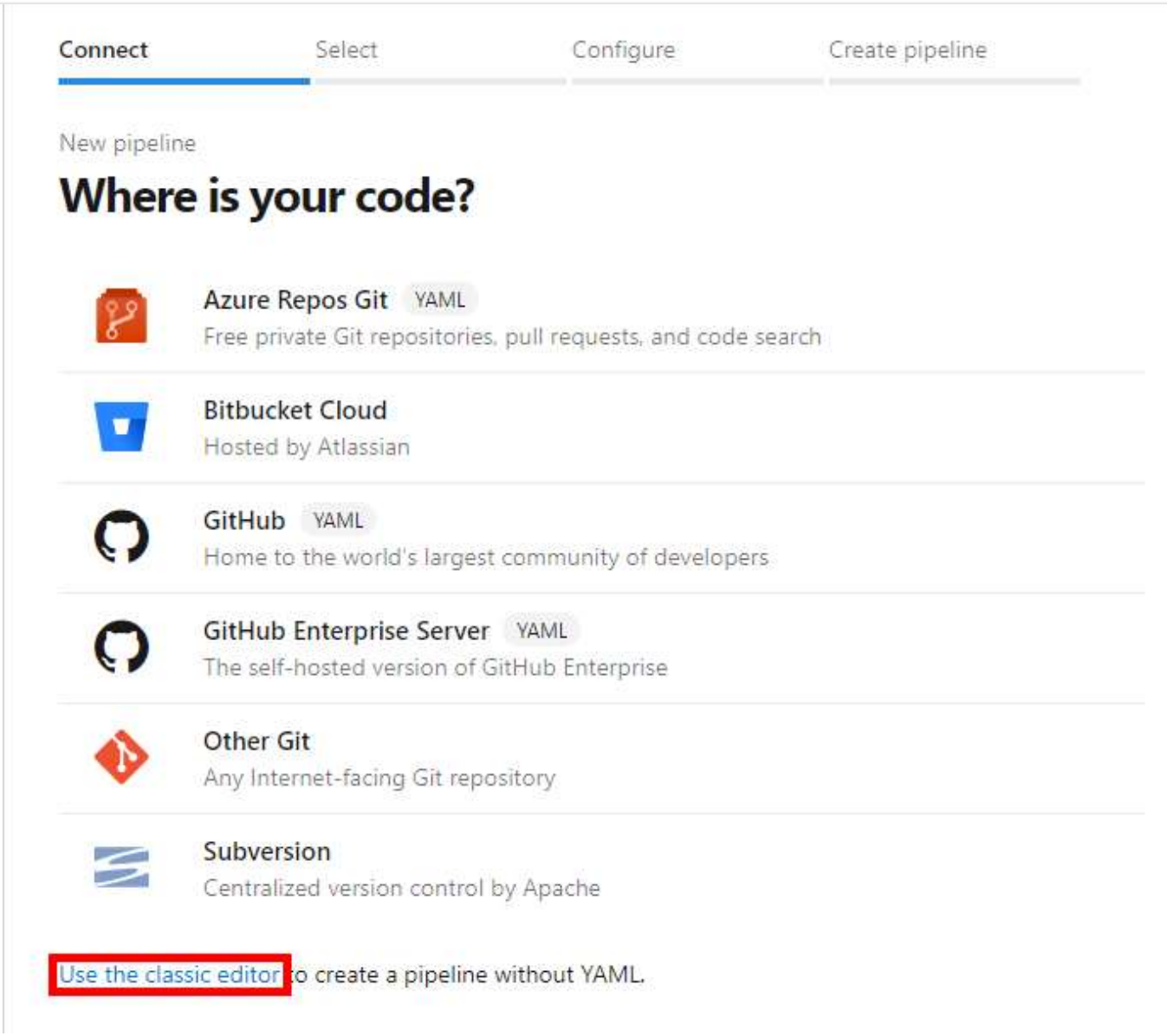


Take a moment to explore the repository. You'll see a **.yml** file representing our build pipeline definition, and a couple of PowerShell scripts.

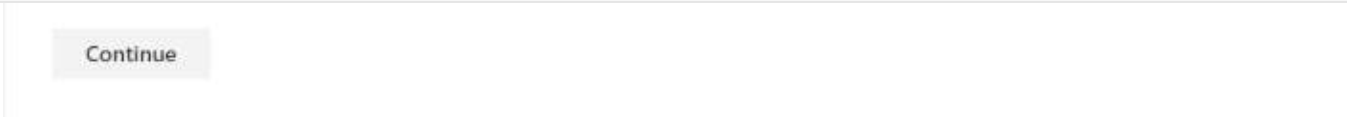
Create the build pipeline

In your Azure DevOps project, open **Pipeline**, then **Builds**.

Select **New**, then **New Build Pipeline**.



Select the repository you created and click **Continue.**



Select the **YAML** template.

Name your pipeline. Select the **Hosted Ubuntu 1604** agent pool and enter a YAML file path of **azure-pipelines.yml**.

Next, we need to configure the pipeline variables. Switch to the **Variables** tab of the pipeline and add the following variables:

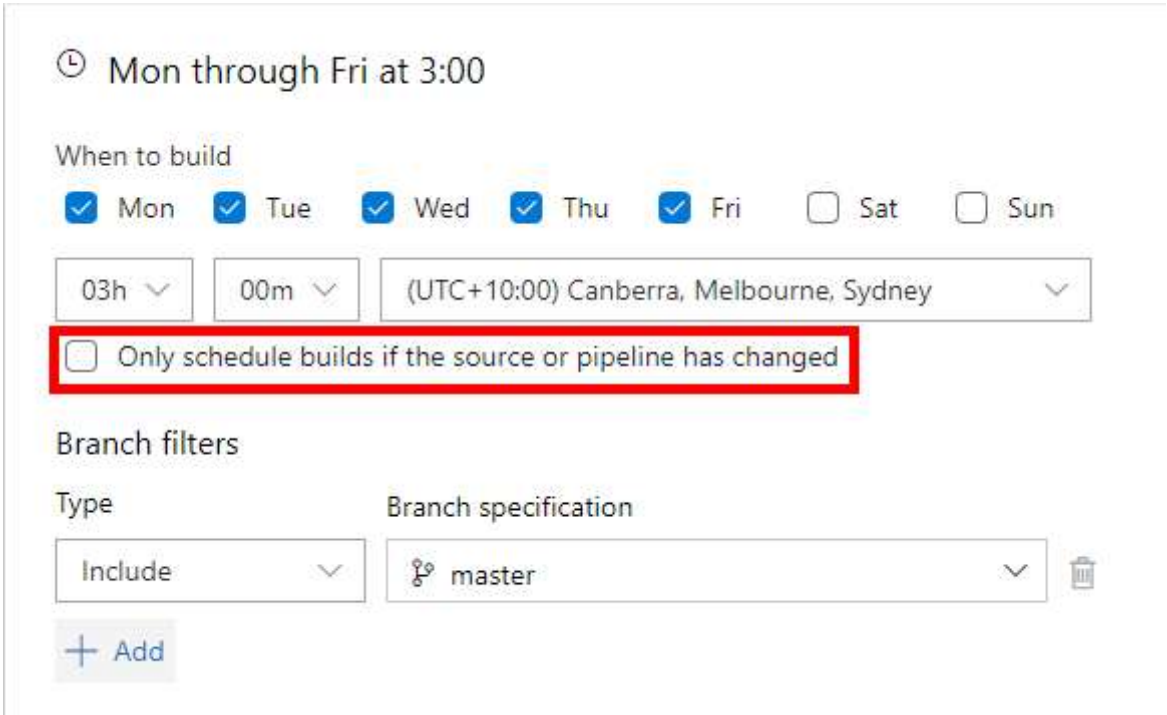
· **AcmeContact**: The email address which should be notified by Let's Encrypt before a certificate is to expire.

· **AcmeDirectory**: The URL of the Acme Directory. Posh-ACME supports a shorthand format for Let's Encrypt. Use "LE_STAGE" for Let's Encrypt staging and "LE_PROD" for Let's Encrypt production. It's best to start with staging and switch to production when ready. Production has strict API limits.

· **CertificateNames**: A comma-separated list of domain names to include on the certificate. The first is the certificate subject. The **Subject Alternate Name** attribute on the certificate consists of any additional names.

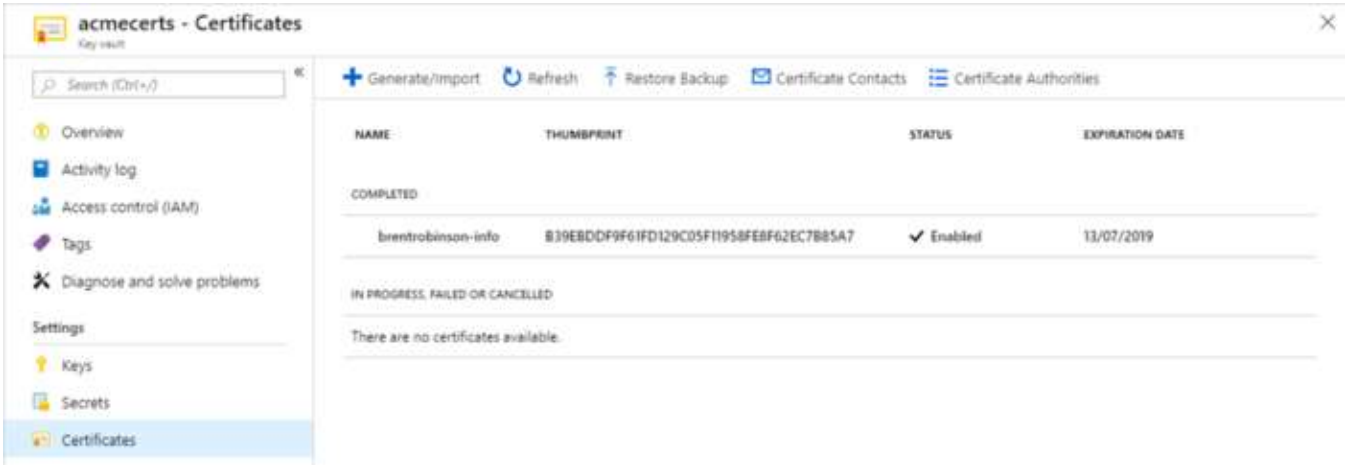
· **StorageContainerSASToken**: The SAS token for the storage container you generated earlier. As this is sensitive, mark it as a secret (which obscures it on the screen, and in logs).

Finally, we want this pipeline to automatically run every day, and renew the certificate if it's ready. Switch to the **Triggers** tab and add a daily schedule. Ensure you disable the option **Only schedule builds if the source or pipeline has changed** as we want it always to run. You could run this pipeline less frequently, such as once a week. Ensure you allow it to run often enough to renew the certificate before it expires. By default, Posh-ACME will only renew the certificate once it's due to expire within 30 days. Technically, you could run it once a month, but you run the risk of a failure leaving your certificate to expire. It's safest to run too frequently as Posh-ACME will skip renewing a certificate which isn't ready.



when called from the scheduled trigger.

The job should take approximately 5 minutes to run. When complete, the certificate will be in your Key Vault.

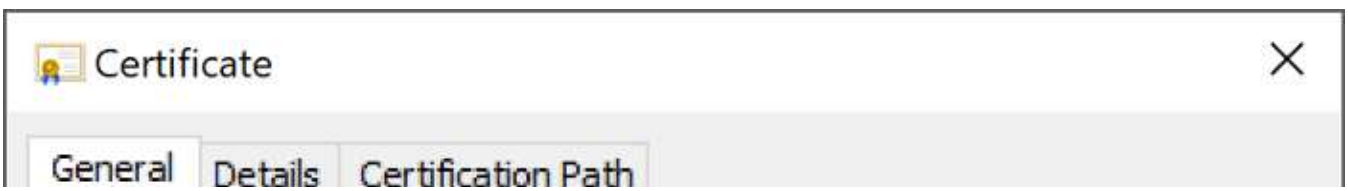


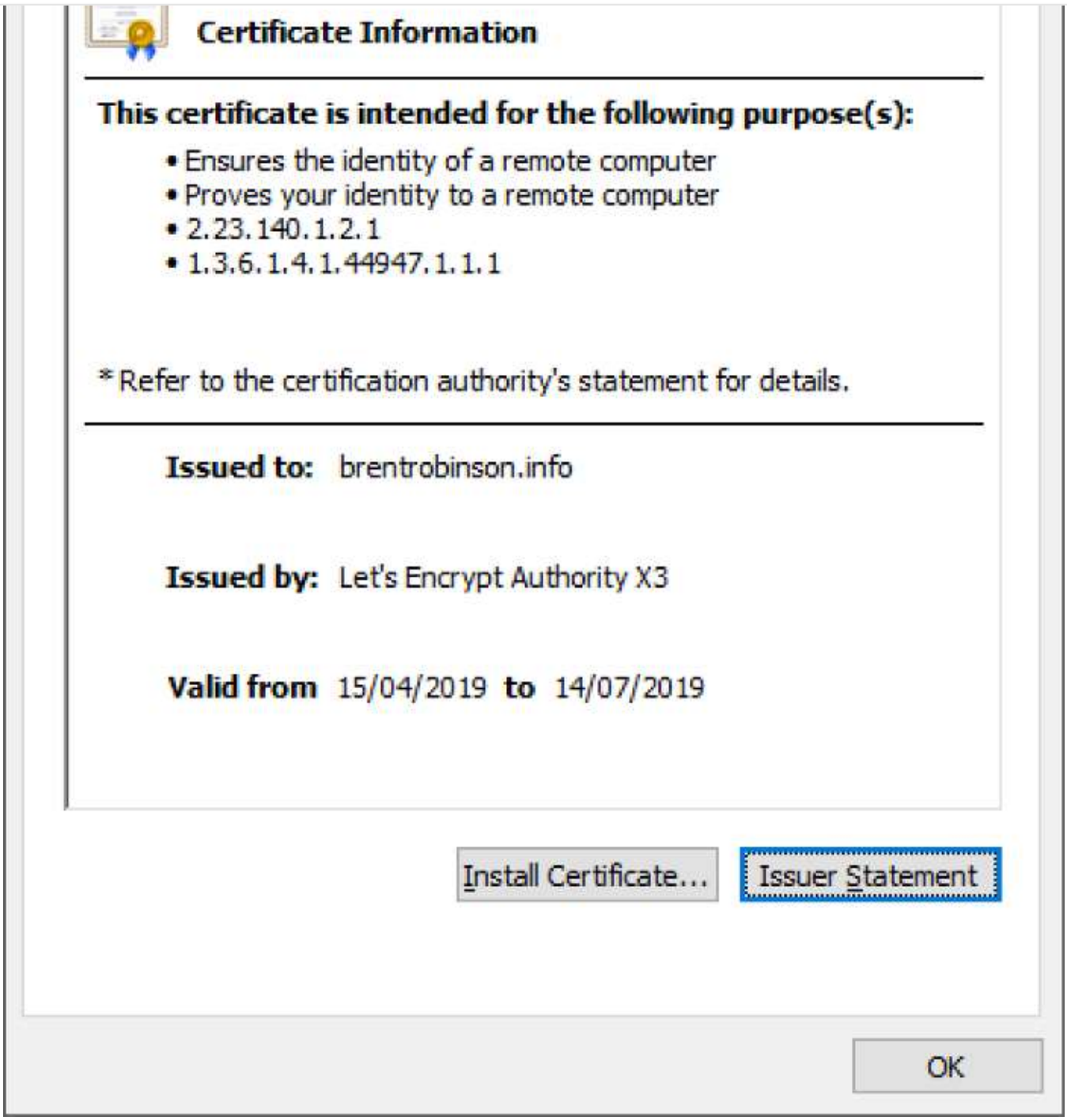
When the certificate is due to expire within 30 days, the pipeline job will automatically renew the certificate and upload the new certificate into your Key Vault.

The certificate

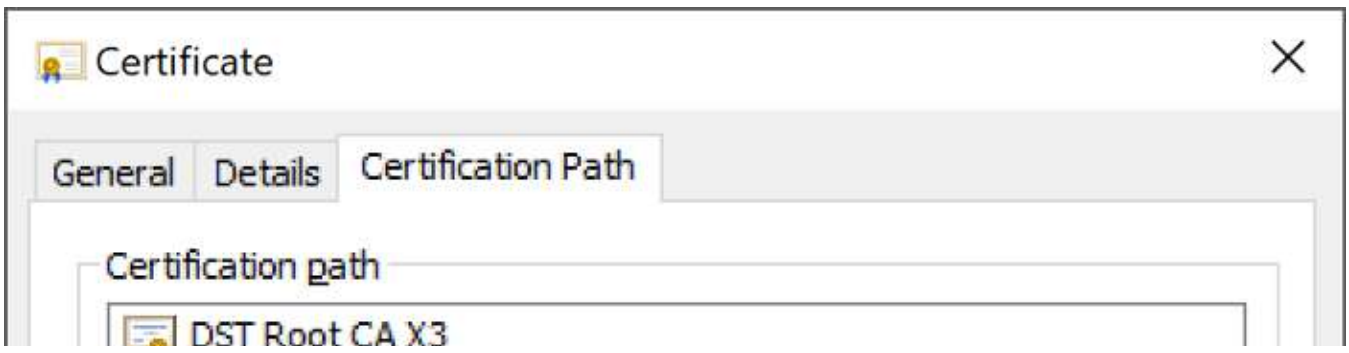
If you've started by using the Let's Encrypt staging environment, the certificate issued won't be trusted. When you've proved the process, switch to the Let's Encrypt production environment which issues trusted certificates. You can do this by updating the **AcmeDirectory** parameter in your pipeline and re-running the pipeline.

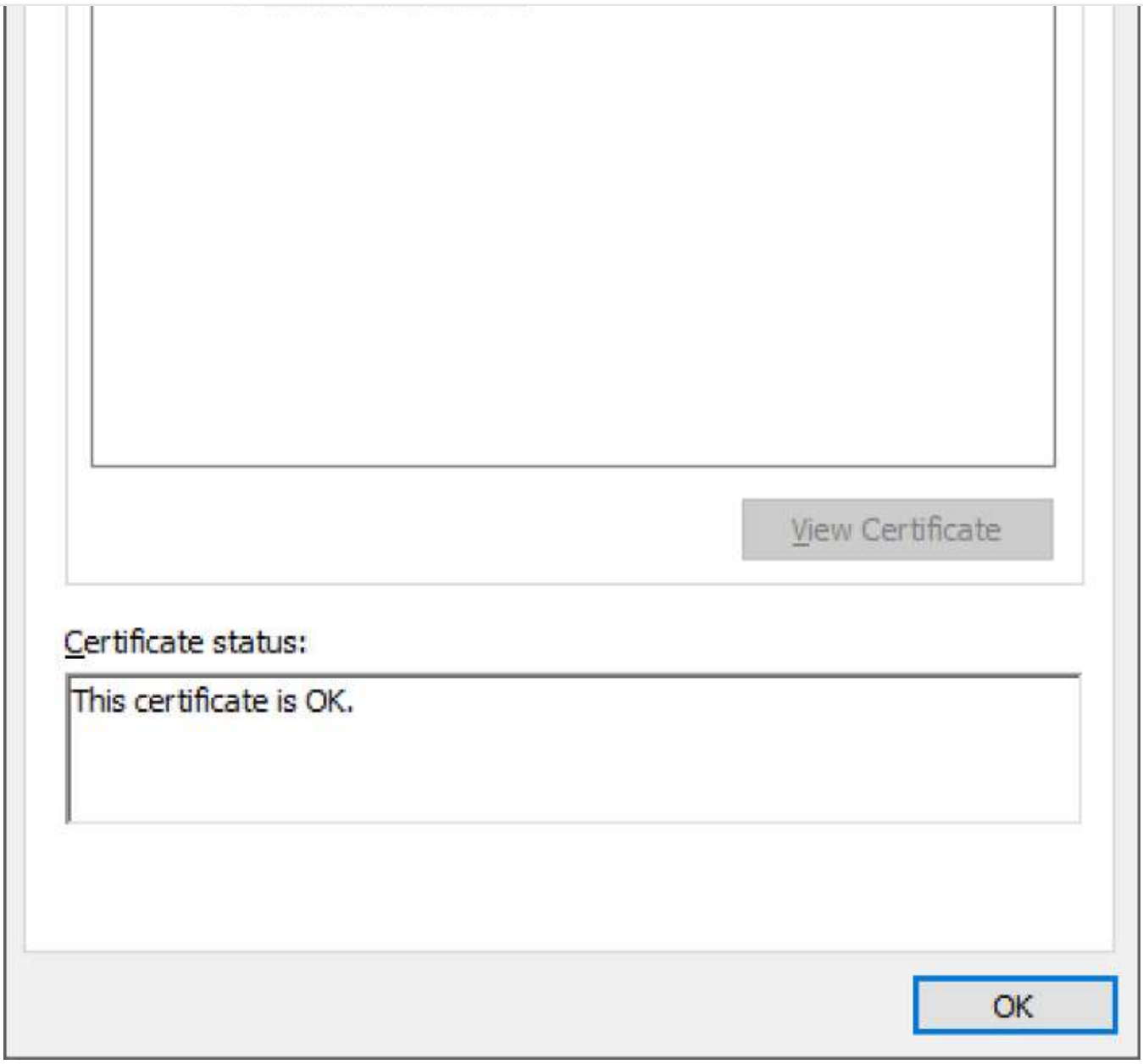
You can download the certificate from Key Vault and inspect it. It looks like any other certificate.





The root is the certificate chain is **DST Root CA X3**, which all major browsers and operating systems trust. *Note: this root may change for certificates issued in the future.*





If you acquired your certificate from Let's Encrypt production, it's ready to use.

Alternatives and limitations

Multiple domains and certificates

You can acquire a certificate for multiple domains by including them in the build pipeline **CertificateNames** parameter. The single certificate issued lists the domains in the **Subject Alternate Name (SAN)** attribute of the certificate.

Multiple certificates are suitable when the certificates are for quite distinct purposes.

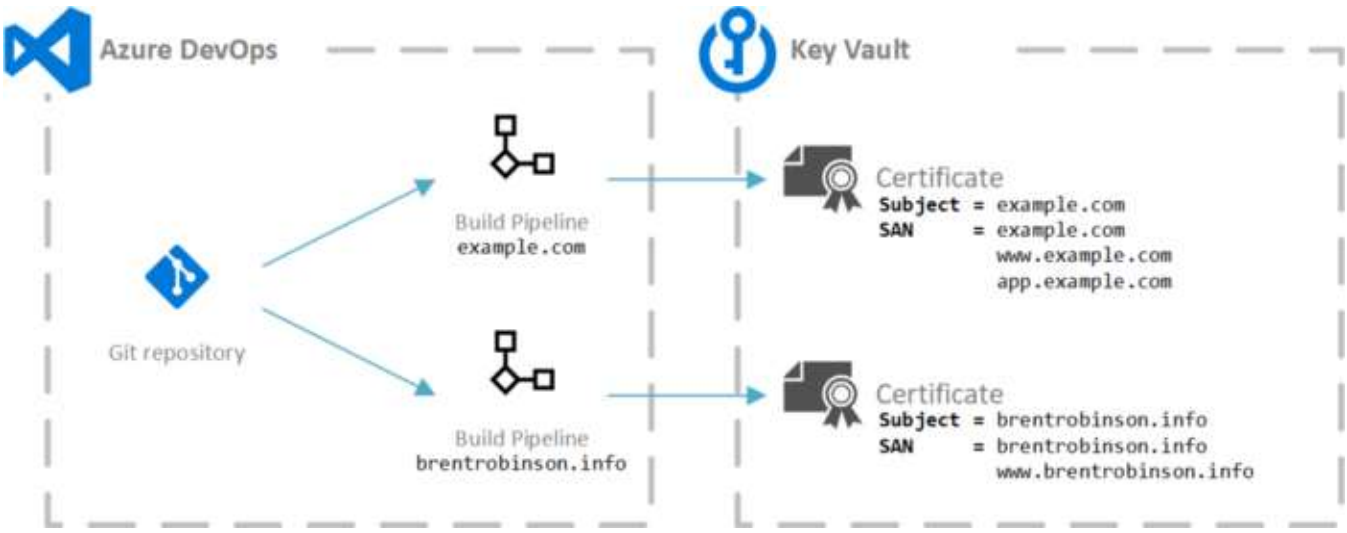


Figure 4: Example of multiple certificates issued

Alternatives to Azure DNS

ACME is in no way specific to Azure DNS. Posh-ACME supports over 25 DNS providers to perform domain validation, and the ACME protocol is DNS provider agnostic. The PowerShell scripts can be modified to connect to an alternate DNS provider for validation. If your provider isn't available in the Posh-ACME list, or for other reasons automating the provisioning of records in the provider isn't feasible (e.g. organisational policy), the ACME approach to certificate issuance can still be used. However, some initial manual setup is required to achieve this.

When validating a domain, the name of the DNS record is predictable. If you're validating the domain "www.example.com", the challenge is the domain prepended with "_acmechallenge", i.e. "_acmechallenge.www.example.com". What's noteworthy of this, is the ACME server, the certificate authority, follows CNAMEs to find the ACME challenge. Therefore, you can point "_acmechallenge.www.example.com" to any DNS location you can control through automation. This alternate DNS location is used to answer all future ACME challenges.

Figure 5: CNAME record to resolve ACME challenges to an alternate provider

Neither DNS provider needs to be Azure DNS. You'll need to create the CNAME records ahead of time, manually; meaning you'll need to know the domains you wish to acquire a certificate for before you start using ACME.

The PowerShell script can be modified to this to let Posh-ACME know in which alternative domain it should be publishing the challenge. Posh-ACME won't detect and follow CNAMEs; you need to be explicit. Only the ACME certificate authority follows the CNAMEs to find the TXT record.

Managed Identity

If you're using private build agents in Azure DevOps, you may wish to leverage a Managed Identity instead of a service principal with a password credential. If you use a Managed Identity, you'll need to modify the PowerShell script to acquire a token through the Identity endpoint.

```
$httpResponse = Invoke-WebRequest -Uri
'http://169.254.169.254/metadata/identity/oauth2/token?api-
version=2018-02-
01&resource=https%3A%2F%2Fmanagement.core.windows.net%2F' -Headers
@{Metadata = "true"}

$azureAccessToken = ($httpResponse.Content | ConvertFrom-
Json).access_token
```

You'll need to ensure your Managed Identity has the same permissions we granted to our service principal.

Windows Support

All the tools we've used are cross-platform, so you could also acquire certificates from Windows build agents. The only change required is to download and install a Windows binary of AzCopy. As we use the AzCopy sync command, at least version 10 of AzCopy is required. At the time of writing, the Windows-based hosted build agents run an older version of AzCopy.

scenario, we only use a single ACME contact and update that contact if it changes. It's possible to pass multiple contacts to Posh-ACME which will configure them on the ACME server. Support for multiple contacts will require some modification to the PowerShell script for the initial contact configuration, and when the contact changes.

Multiple Azure subscriptions

The Azure PowerShell task we use in the Azure DevOps pipeline automatically connects to Azure and selects a subscription based on what was selected when setting up the service principal. As such, the PowerShell script executes against a single subscription. If your DNS zones are spread among multiple subscriptions, the script will fail to find them. The PowerShell script can be modified to support discovering DNS zones across multiple subscriptions, but be aware, you must tell Posh-ACME which subscription it can find each DNS zone to publish the challenge.

Other Certificate Authorities

Let's Encrypt isn't the only ACME compatible certificate authority. It's still early days for ACME, but its adoption rate is growing. ACME is a standard, so you can switch to any ACME compatible certificate authority. There are quirks of some authorities, but much of the ACME tools and libraries are tolerant of these. Though in most cases, Let's Encrypt should meet your needs.

Azure TIs Azure Devops Acme Lets Encrypt