Tell us about your PDF experience.

Azure and GitHub integration

Learn how GitHub and Azure work together to let you build and deploy apps.

GitHub Actions for Azure

GET STARTED

What is GitHub Actions for Azure?

Deploy to Azure

DEPLOY

Deploy apps from GitHub to Azure

Deploy databases from GitHub to Azure

Build custom virtual images

Tools for interacting with GitHub Actions

GET STARTED

Authenticate from Azure to GitHub

Use variable substitution

Azure Developer CLI (azd) Preview

DEPLOY

What is Azure Developer CLI?

Get started

See more in the Azure Developer CLI developer center

Azure Pipelines and GitHub integration

HOW-TO GUIDE

Work with Azure DevOps and GitHub

Manage Azure Policies with GitHub

GET STARTED

Manage Azure Policies as code with GitHub

What is GitHub Actions for Azure

Article • 12/16/2022 • 3 minutes to read

GitHub Actions helps you automate your software development workflows from within GitHub. You can deploy workflows in the same place where you store code and collaborate on pull requests and issues.

In GitHub Actions, a workflow is an automated process that you set up in your GitHub repository. You can build, test, package, release, or deploy any project on GitHub with a workflow.

Each workflow is made up of individual actions that run after a specific event (like a pull request) occur. The individual actions are packaged scripts that automate software development tasks.

With GitHub Actions for Azure, you can create workflows that you can set up in your repository to build, test, package, release, and deploy to Azure. GitHub Actions for Azure supports Azure services, including Azure App Service, Azure Functions, and Azure Key Vault.

GitHub Actions also include support for utilities, including Azure Resource Manager templates, Azure CLI, and Azure Policy.

Watch this video from GitHub Universe 2020 to learn more about continuous delivery with GitHub Actions.
https://www.youtube-nocookie.com/embed/36hY0-O4STg

Why should I use GitHub Actions for Azure

GitHub Actions for Azure are developed by Microsoft and designed to be used with Azure. You can see all of the GitHub Actions for Azure in the GitHub Marketplace. See Finding and customizing actions to learn more about incorporating actions into your workflows.

What is the difference between GitHub Actions and Azure Pipelines

Azure Pipelines and GitHub Actions both help you automate software development workflows. Learn more about how the services differ and how to migrate from Azure Pipelines to GitHub Actions.

What do I need to use GitHub Actions for Azure

You'll need Azure and GitHub accounts:

- An Azure account with an active subscription. Create an account for free.
- A GitHub account. If you don't have one, sign up for free.

How do I connect GitHub Actions and Azure

Depending on the action, you'll use a service principal or publish profile to connect to Azure from GitHub. You'll use a service principal each time you use the Azure login action. The Azure App Service action supports using a publish profile or service principal. See Application and service principal objects in Azure Active Directory to learn more about service principals.

You can use the Azure login action in combination with both the Azure CLI and Azure Azure PowerShell actions. The Azure login action also works with most other GitHub actions for Azure including deploying to web apps and accessing key vault secrets.

What is included in a GitHub Actions workflow

Workflows are made up of one or more jobs. Within a job, there are steps made up of individual actions. See Introduction to GitHub Actions to learn more about GitHub Actions concepts.

Where can I see complete workflow examples

The Azure starter action workflows repository includes end-to-end workflows to build and deploy Web apps of any language, any ecosystem to Azure.

Where can I see all the available actions

Visit the Marketplace for GitHub Actions for Azure to see all the available GitHub Actions for Azure.

- Deploy Bicep file or Azure Resource Manager template
- Deploy to a static web app
- Azure App Service settings
- Deploy to Azure Functions

- Deploy to Azure Functions for Containers
- Docker login
- Deploy to Azure Container Instances
- Container scanning action
- Kubectl tool installer
- Kubernetes set context
- AKS set context
- Kubernetes create secret
- Kubernetes deploy
- Setup Helm
- Kubernetes bake
- Build Azure virtual machine images
- Machine learning login
- Machine learning training
- Machine learning - deploy model
- Deploy to Azure SQL database
- Deploy to Azure MySQL action
- Azure Policy Compliance Scan
- Manage Azure Policy
- Trigger an Azure Pipelines run
- Variable substitution

Next Steps

Learning path, Automate your workflow with GitHub Actions

Learning Lab, Continuous Delivery with Azure

Use GitHub Actions to connect to Azure

Article • 10/26/2022 • 10 minutes to read

Learn how to use Azure login with either Azure PowerShell or Azure CLI to interact with your Azure resources.

To use Azure PowerShell or Azure CLI in a GitHub Actions workflow, you need to first log in with the Azure login action.

The Azure login action supports two different ways of authenticating with Azure:

- Service principal with secrets
- OpenID Connect (OIDC) with a Azure service principal using a Federated Identity Credential

By default, the login action logs in with the Azure CLI and sets up the GitHub action runner environment for Azure CLI. You can use Azure PowerShell with `enable-AzPSSession` property of the Azure login action. This sets up the GitHub action runner environment with the Azure PowerShell module.

You can use Azure login to connect to public or sovereign clouds including Azure Government and Azure Stack Hub.

Use the Azure login action with OpenID Connect

To set up an Azure Login with OpenID Connect and use it in a GitHub Actions workflow, you'll need:

- An Azure Active Directory application, with a service principal that has contributor access to your subscription
- An Azure Active Directory application configured with a federated credential to trust tokens issued by GitHub Actions to your GitHub repository. You can configure this in the Azure portal or with Microsoft Graph REST APIs
- A GitHub Actions workflow that requests GitHub issue tokens to the workflow, and uses the Azure login action

Create an Azure Active Directory application and service principal

You'll need to create an Azure Active Directory application and service principal and then assign a role on your subscription to your application so that your workflow has access to your subscription.

Azure portal

1. If you do not have an existing application, register a new Azure Active Directory application and service principal that can access resources. As part of this process, make sure to:

   - Register your application with Azure AD and create a service principal
   - Assign a role to the application

2. Open **App registrations** in Azure portal and find your application. Copy the values for **Application (client) ID** and **Directory (tenant) ID** to use in your GitHub Actions workflow.

3. Open **Subscriptions** in Azure portal and find your subscription. Copy the **Subscription ID**.

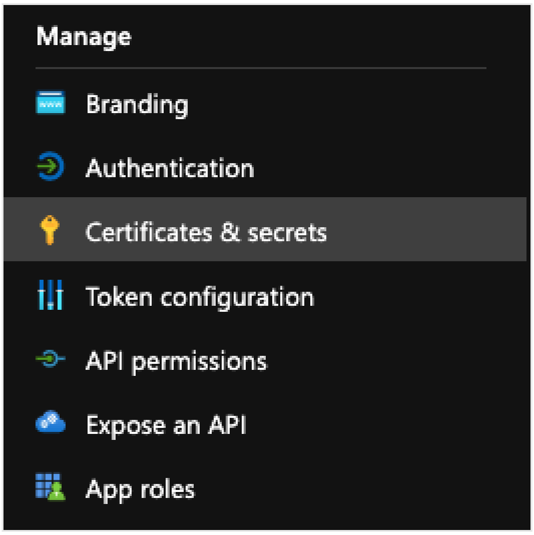
Add federated credentials

You can add federated credentials in the Azure portal or with the Microsoft Graph REST API.

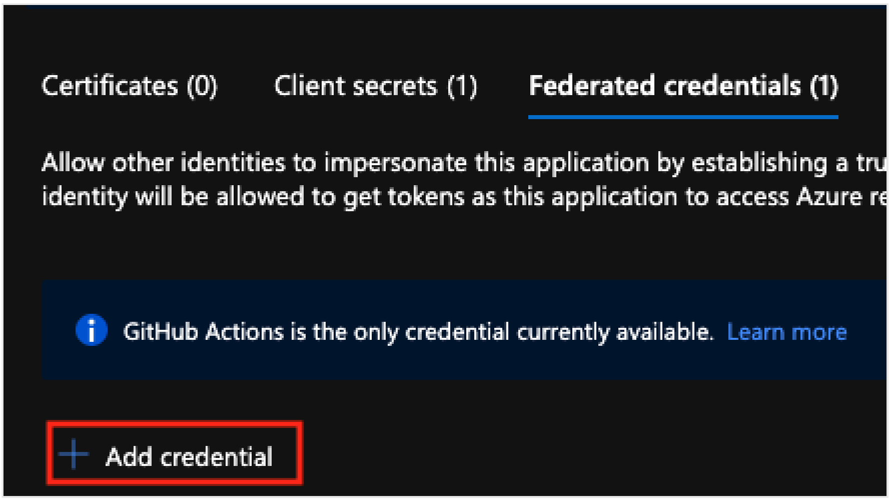
Azure portal

1. Go to **App registrations** in the Azure portal and open the app you want to configure.

2. Within the app, go to **Certificates and secrets**.



3. In the **Federated credentials** tab, select **Add credential**.



Allow other identities to impersonate this application by establishing a tru
identity will be allowed to get tokens as this application to access Azure re

GitHub Actions is the only credential currently available.

4. Select the credential scenario **GitHub Actions deploying Azure resources**. Generate your credential by entering your credential details.

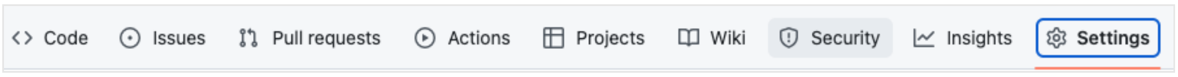
Field	Description	Example
Organization	Your GitHub organization name or GitHub username.	`contoso`
Repository	Your GitHub Repository name.	`contoso-app`
Entity type	The filter used to scope the OIDC requests from GitHub workflows. This field is used to generate the `subject` claim.	`Environment`, `Branch`, `Pull request`, `Tag`
GitHub name	The name of the environment, branch, or tag.	`main`
Name	Identifier for the federated credential.	`contoso-deploy`

For a more detailed overview, see Configure an app to trust a GitHub repo.

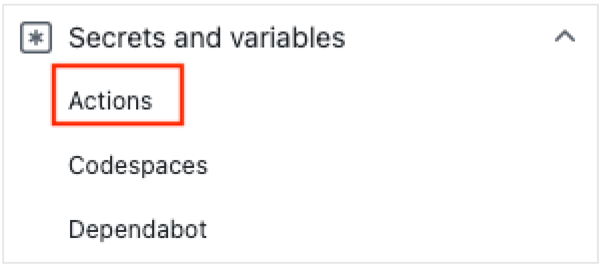
Create GitHub secrets

You need to provide your application's **Client ID**, **Tenant ID** and **Subscription ID** to the login action. These values can either be provided directly in the workflow or can be stored in GitHub secrets and referenced in your workflow. Saving the values as GitHub secrets is the more secure option.

1. Open your GitHub repository and go to **Settings**.



2. Select **Security > Secrets and variables > Actions**.



3. Create secrets for `AZURE_CLIENT_ID`, `AZURE_TENANT_ID`, and `AZURE_SUBSCRIPTION_ID`. Use these values from your Azure Active Directory application for your GitHub secrets:

GitHub Secret	Azure Active Directory Application
AZURE_CLIENT_ID	Application (client) ID
AZURE_TENANT_ID	Directory (tenant) ID
AZURE_SUBSCRIPTION_ID	Subscription ID

4. Save each secret by selecting **Add secret**.

Set up Azure Login with OpenID Connect authentication

Your GitHub Actions workflow uses OpenID Connect to authenticate with Azure. To learn more about this interaction, see the GitHub Actions documentation.

In this example, you'll use OpenID Connect Azure CLI to authenticate with Azure with the Azure login action. The example uses GitHub secrets for the `client-id`, `tenant-id`, and `subscription-id` values. You can also pass these values directly in the login action.

The Azure login action includes an optional `audience` input parameter that defaults to `api://AzureADTokenExchange`. You can update this parameter for custom audience values.

Windows

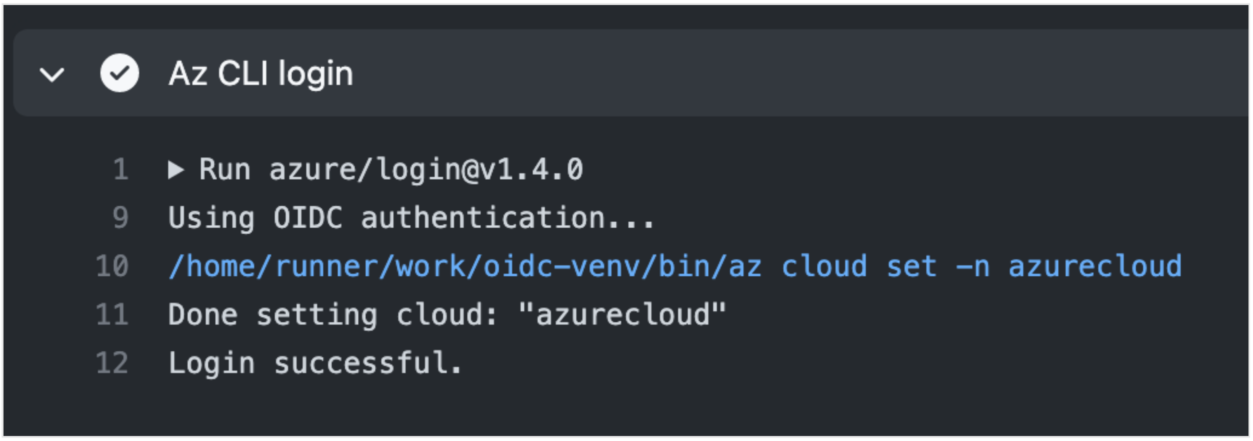
This workflow authenticates with OpenID Connect and uses PowerShell to output a list of resource groups tied to the connected Azure subscription.

YAML

```yaml
name: Run Azure Login with OpenID Connect and PowerShell
on: [push]

permissions:
      id-token: write
      contents: read

jobs:
  Windows-latest:
      runs-on: windows-latest
      steps:
        - name: OIDC Login to Azure Public Cloud with AzPowershell
(enableAzPSSession true)
          uses: azure/login@v1
          with:
            client-id: ${{ secrets.AZURE_CLIENT_ID }}
            tenant-id: ${{ secrets.AZURE_TENANT_ID }}
            subscription-id: ${{ secrets.AZURE_SUBSCRIPTION_ID }}
            enable-AzPSSession: true

        - name: 'Get resource group with PowerShell action'
          uses: azure/powershell@v1
          with:
             inlineScript: |
               Get-AzResourceGroup
             azPSVersion: "latest"
```

Verify successful Azure Login with OpenID

Open the `Az CLI login` action and verify that it ran successfully. You should see the message `Login successful`. If your login is unsuccessful, you'll see the message `Az CLI Login failed.`.



Az CLI login

```
▶ Run azure/login@v1.4.0
Using OIDC authentication...
/home/runner/work/oidc-venv/bin/az cloud set -n azurecloud
Done setting cloud: "azurecloud"
Login successful.
```

Use the Azure login action with a service principal secret

To use Azure login with a service principal, you first need to add your Azure service principal as a secret to your GitHub repository.

Create a service principal

In this example, you will create a secret named `AZURE_CREDENTIALS` that you can use to authenticate with Azure.

1. Open Azure Cloud Shell in the Azure portal or Azure CLI locally.

   Note

   If you are using Azure Stack Hub, you'll need to set your SQL Management endpoint to `not supported`. `az cloud update -n {environmentName} --endpoint-sql-management https://notsupported`

2. Create a new service principal in the Azure portal for your app. The service principal must be assigned the Contributor role.

   Azure CLI

   ```
       az ad sp create-for-rbac --name "myApp" --role contributor \
                               --scopes /subscriptions/{subscription-id}/resourceGroups/{resource-group} \
                               --sdk-auth
   ```

3. Copy the JSON object for your service principal.

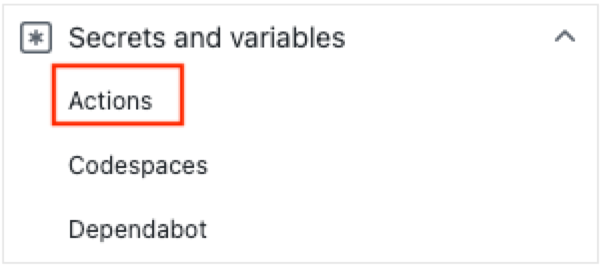
JSON

```json
{
    "clientId": "<GUID>",
    "clientSecret": "<GUID>",
    "subscriptionId": "<GUID>",
    "tenantId": "<GUID>",
    (...)
}
```

Add the service principal as a GitHub secret

1. In GitHub, go to your repository.

2. Select **Security** > **Secrets and variables** > **Actions**.



3. Select **New repository secret**.

4. Paste the entire JSON output from the Azure CLI command into the secret's value field. Give the secret the name `AZURE_CREDENTIALS`.

5. Select **Add secret**.

Use the Azure login action

Use the service principal secret with the Azure Login action to authenticate to Azure.

In this workflow, you authenticate using the Azure login action with the service principal details stored in `secrets.AZURE_CREDENTIALS`. Then, you run an Azure CLI action. For more information about referencing GitHub secrets in a workflow file, see Using encrypted secrets in a workflow in GitHub Docs.

Once you have a working Azure login step, you can use the Azure PowerShell or Azure CLI actions. You can also use other Azure actions, like Azure webapp deploy and Azure functions.

YAML

```
on: [push]

name: AzureLoginSample

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest
    steps:
      - name: Log in with Azure
        uses: azure/login@v1
        with:
          creds: '${{ secrets.AZURE_CREDENTIALS }}'
```

Use the Azure PowerShell action

In this example, you log in with the Azure Login action and then retrieve a resource group with the Azure PowerShell action.

YAML

```yaml
on: [push]

name: AzureLoginSample

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest
    steps:
      - name: Log in with Azure
        uses: azure/login@v1
        with:
          creds: '${{ secrets.AZURE_CREDENTIALS }}'
          enable-AzPSSession: true
      - name: Azure PowerShell Action
        uses: Azure/powershell@v1
        with:
          inlineScript: Get-AzResourceGroup -Name "< YOUR RESOURCE GROUP >"
          azPSVersion: "latest"
```

Use the Azure CLI action

In this example, you log in with the Azure Login action and then retrieve a resource group with the Azure CLI action.

YAML

```yaml
on: [push]
```

```
name: AzureLoginSample

jobs:
build-and-deploy:
    runs-on: ubuntu-latest
    steps:

    - name: Log in with Azure
        uses: azure/login@v1
        with:
        creds: ${{ secrets.AZURE_CREDENTIALS }}

    - name: Azure CLI script
        uses: azure/CLI@v1
        with:
        azcliversion: 2.0.72
        inlineScript: |
            az account show
            az storage -h
```

Connect to Azure Government and Azure Stack Hub clouds

To log in to one of the Azure Government clouds, set the optional parameter environment with supported cloud names `AzureUSGovernment` or `AzureChinaCloud`. If this parameter is not specified, it takes the default value `AzureCloud` and connects to the Azure Public Cloud.

YAML

```yaml
   - name: Login to Azure US Gov Cloud with CLI
     uses: azure/login@v1
        with:
          creds: ${{ secrets.AZURE_US_GOV_CREDENTIALS }}
          environment: 'AzureUSGovernment'
          enable-AzPSSession: false
   - name: Login to Azure US Gov Cloud with Az Powershell
      uses: azure/login@v1
        with:
          creds: ${{ secrets.AZURE_US_GOV_CREDENTIALS }}
          environment: 'AzureUSGovernment'
          enable-AzPSSession: true
```

Connect with other Azure services

The following articles provide details on connecting to GitHub from Azure and other services.

Azure Active Directory

- Sign in to GitHub Enterprise with Azure AD (single sign-on)

Power BI

- Connect Power BI with GitHub

Connectors

- GitHub connector for Azure Logic Apps, Power Automate and Power Apps

Azure Databricks

- Use GitHub as version control for notebooks

Deploy apps from GitHub to Azure

Deploy apps from GitHub to Azure

Article • 09/22/2022 • 2 minutes to read

The following articles provide support to deploy apps from GitHub to Azure.

Azure App Service

- Deploy to Azure App Service on Linux using GitHub Actions
- Deploy an Azure App Service Custom Container with GitHub Actions
- Deploy to App Service on Linux and connect to a database
- Deploy to Azure App Service on Linux using Visual Studio Code
- Tutorial: Use GitHub Actions to deploy to an App Service Custom Container and connect to a database

Azure Functions

- Deploy a function app continuously from GitHub
- Deploy to Azure Functions using GitHub Actions

Azure App Configuration

- Sync your GitHub repository to App Configuration

Azure Key Vault

- Use Key Vault secrets in GitHub Actions workflows

Azure Storage

- Use GitHub Actions workflow to deploy your static website in Azure Storage

Azure Container Instances

- Configure a GitHub action to create a container instance

Azure Container Registry

- Scan container images using GitHub Actions

Azure Kubernetes Service

- Use GitHub Actions to deploy to Kubernetes
- Deploy to Azure Dev Spaces using GitHub Actions

Azure Shared Image Gallery

- Build custom virtual machine images with GitHub Actions

Azure Pipelines

- Trigger a Pipeline run from GitHub Actions

Azure Resource Manager templates

- Deploy Bicep files by using GitHub Actions
- Deploy Azure Resource Manager templates by using GitHub Actions

Azure Machine Learning

- Use GitHub Actions with Azure Machine Learning

Azure Stack

- Use the Azure login action with Azure CLI and PowerShell on Azure Stack Hub

Deploy databases from GitHub to Azure

Article • 09/22/2022 • 2 minutes to read

The following articles provide support to deploy database updates from GitHub to Azure. You can use GitHub Actions to deploy to Azure SQL, Azure MySQL, and Azure Database for PostgreSQL.

- Use GitHub Actions to connect to Azure SQL Database
- Use GitHub Actions to connect to Azure MySQL
- Use GitHub Actions to connect to Azure PostgreSQL

Use variable substitution with GitHub Actions

Article • 10/26/2022 • 2 minutes to read

Learn how to use variable substitution action to replace values in XML, JSON and YAML based configuration and parameter files.

Variable substitution lets you insert values, including GitHub secrets, into files in your repository during the workflow run. For example, you could insert an API login and password into a JSON file during the workflow run.

Variable substitution only works for keys predefined in the object hierarchy. You cannot create new keys with variable substitution. In addition, only variables defined as environment variables in the workflow or system variables that are already available can be used for substitution.

Prerequisites

- A GitHub account. If you don't have one, sign up for free.

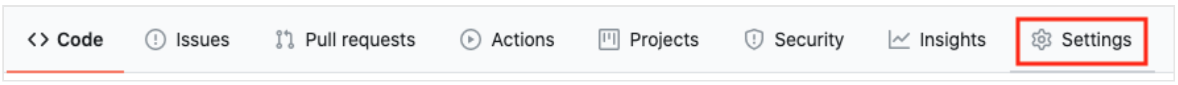
Use the variable substitution action

This example walks through replacing values in `employee.json` using the variable substitution action.

1. Create `employee.json` at the root level of your repository.

```JSON
{
    "first-name": "Toni",
    "last-name": "Cranz",
    "username": "",
    "password": "",
    "url": ""
}
```

2. Open your GitHub repository and go to **Settings**.



3. Select **Security** > **Secrets and variables** > **Actions**.

4. Select **New repository secret**.

5. Add a new secret `PASSWORD` with the value `5v{W<$2B<GR2=t4#` (or a password you select). Save your secret.

6. Go to **Actions** and select **set up a workflow yourself**.

7. Add a workflow file. The username value in your json file will be replaced with `tcranz`. The password will be replaced with your GitHub secret. The url field will be populated with a URL that includes the GitHub variable `github.repository`.

   YAML

   ```yaml
   on: [push]
   name: variable substitution in json

   jobs:
   build:
       runs-on: ubuntu-latest
       steps:
       - uses: actions/checkout@v2
       - uses: microsoft/variable-substitution@v1
       with:
           files: 'employee.json'
       env:
           username: tcranz
           password: ${{ secrets.PASSWORD }}
           url: https://github.com/${{github.repository}}
   ```

8. Go to **Actions** to see your workflow run. Open the variable substitution action. You should see that each variable was replaced.

   text

   ```text
   SubstitutingValueonKeyWithString username tcranz
   SubstitutingValueonKeyWithString password ***
   SubstitutingValueonKeyWithString url
   https://github.com/account/variable-sub
   Successfully updated file: employee.json
   ```

Clean up resources

Delete your GitHub repository when it is no longer needed.

Next steps

Deploy to Azure Web Apps using GitHub Actions

Use Key Vault secrets in GitHub Actions workflows

Article • 10/26/2022 • 4 minutes to read

ⓘ **Note**

The **Azure Key Vault action** is deprecated. The recommended alternative is to use the **Azure CLI action** and pass a custom script to access Azure Key Vault.

Use Key Vault secrets in your GitHub Actions and securely store passwords and other secrets in an Azure Key Vault. Learn more about Key Vault.

Key Vault secrets differ from GitHub secrets:

- Key Vault lets you centralize storage of application secrets in Azure. GitHub secrets are stored in GitHub
- Key Vault can be used as a key and certificate management solutions, in addition to a tool for secrets management
- Key Vault uses Azure role-based access control (Azure RBAC) for access

When you combine Key Vault and GitHub Actions, you have the benefits of a centralized secrets management tool and all the advantages of GitHub Actions.

Prerequisites

- A GitHub account. If you don't have one, sign up for free.
- An Azure account with an active subscription. Create an account for free.
- An Azure App connected to a GitHub repository. This example uses Deploy containers to Azure App Service.
- A Key Vault. You can create a Key Vault using the Azure portal, Azure CLI, or Azure PowerShell.

Workflow file overview

The YAML workflow file includes two sections:

Section	Tasks

Section	Tasks
Authentication	1. Define a service principal. 2. Create a GitHub secret. 3. Add a role assignment.
Key Vault	1. Add the key vault action. 2. Reference the key vault secret.

Learn more about the components of GitHub Actions.

Define a service principal

You can create a service principal with the az ad sp create-for-rbac command in the Azure CLI. Run this command with Azure Cloud Shell in the Azure portal or by selecting the **Try it** button.

Azure CLI

```
   az ad sp create-for-rbac --name {myApp} --role contributor --scopes
/subscriptions/{subscription-id}/resourceGroups/{MyResourceGroup} --sdk-auth
```

In the example above, replace the placeholders with your subscription ID and resource group name. Replace the placeholder `myApp` with the name of your application. The output is a JSON object with the role assignment credentials that provide access to your App Service app similar to below. Copy this JSON object for later. You can shorten the JSON object to only include the lines with the `clientId`, `clientSecret`, `subscriptionId`, and `tenantId` values.

Output

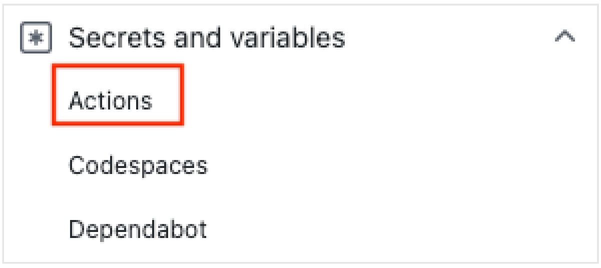
```
  {
    "clientId": "<GUID>",
    "clientSecret": "<GUID>",
    "subscriptionId": "<GUID>",
    "tenantId": "<GUID>",
    (...)
  }
```

Create a GitHub secret

Create secrets for your Azure credentials, resource group, and subscriptions.

1. In GitHub, go to your repository.

2. Select **Security** > **Secrets and variables** > **Actions**.



3. Select **New repository secret**.

4. Paste the entire JSON output from the Azure CLI command into the secret's value field. Give the secret the name `AZURE_CREDENTIALS`.

5. Select **Add secret**.

Add a role assignment

Grant access to the Azure service principal so that you can access your key vault for `get` and `list` operations. If you don't do this, then you will not be able to use the service principal.

Replace `keyVaultName` with the name of your key vault and `clientIdGUID` with the value of your `clientId`.

Azure CLI

```
    az keyvault set-policy -n {keyVaultName} --secret-permissions get list --spn {clientIdGUID}
```

Add the key vault action

With the Azure Key Vault action, you can fetch one or more secrets from a key vault instance and consume it in your GitHub Actions workflows.

Secrets fetched are set as outputs and also as environment variables. Variables are automatically masked when they are printed to the console or to logs.

YAML

```yaml
    - uses: Azure/get-keyvault-secrets@v1
      with:
        keyvault: "my Vault" # name of key vault in Azure portal
        secrets: 'mySecret'  # comma separated list of secret keys to fetch
```

```
from key vault
      id: myGetSecretAction # ID for secrets that you will reference
```

Add the Azure Login Action

For GitHub actions that don't use public endpoints, you may need to configure the Azure Login Action.

Reference the key vault secret

To use a key vault in your workflow, you need both the key vault action and to reference that action.

In this example, the key vault is named `containervault`. Two key vault secrets are added to the environment with the key vault action - `containerPassword` and `containerUsername`.

The key vault values are later referenced in the docker login task with the prefix `steps.myGetSecretAction.outputs`. For example, the username value is referenced as `${{ steps.myGetSecretAction.outputs.containerUsername }}`.

The syntax for referencing GitHub secret is different. In the checkout action, the `AZURE_CREDENTIALS` secret is referenced with `${{ secrets.AZURE_CREDENTIALS }}`.

YAML

```yaml
name: Example key vault flow

on: [push]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
    # checkout the repo
    - uses: actions/checkout@v2
    - uses: Azure/login@v1
      with:
        creds: ${{ secrets.AZURE_CREDENTIALS }}
    - uses: Azure/get-keyvault-secrets@v1
      with:
        keyvault: "containervault"
        secrets: 'containerPassword, containerUsername'
      id: myGetSecretAction
    - uses: azure/docker-login@v1
      with:
        login-server: myregistry.azurecr.io
        username: ${{ steps.myGetSecretAction.outputs.containerUsername }}
        password: ${{ steps.myGetSecretAction.outputs.containerPassword }}
    - run: |
        docker build . -t myregistry.azurecr.io/myapp:${{ github.sha }}
        docker push myregistry.azurecr.io/myapp:${{ github.sha }}
    - uses: azure/webapps-deploy@v2
      with:
        app-name: 'myapp'
        publish-profile: ${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE }}
        images: 'myregistry.azurecr.io/myapp:${{ github.sha }}'
```

Clean up resources

When your Azure app, GitHub repository, and key vault are no longer needed, clean up the resources you deployed by deleting the resource group for the app, GitHub repository, and key vault.

Next steps

Learn more about Key Vault

Manage Azure Policies with GitHub

Article • 09/22/2022 • 2 minutes to read

Review the following articles to learn how to manage Azure Policies as code from GitHub

- Export Azure Policies from Azure
- Manage Azure Policies as code from GitHub
- Trigger Azure compliance scans

Build custom virtual machine images with GitHub Actions and Azure

Article • 10/26/2022 • 10 minutes to read

Get started with the GitHub Actions by creating a workflow to build a virtual machine image.

With GitHub Actions, you can speed up your CI/CD process by creating custom virtual machine images with artifacts from your workflows. You can both build images and distribute them to a Shared Image Gallery.

You can then use these images to create virtual machines and virtual machine scale sets.

The build virtual machine image action uses the Azure Image Builder service.

Prerequisites

- An Azure account with an active subscription. Create an account for free .
- A GitHub account with an active repository. If you don't have one, sign up for free .
  - This example uses the Java Spring PetClinic Sample Application .
- A Shared Image Gallery.
  - Create a Shared Image Gallery with the Azure CLI
  - Create an Azure Shared Image Gallery using the portal (Windows, Linux)

Workflow file overview

A workflow is defined by a YAML (.yml) file in the `/.github/workflows/` path in your repository. This definition contains the various steps and parameters that make up the workflow.

The file has three sections:

Section	Tasks
Authentication	1. Add a user-managed identity. 2. Set up a service principal or Open ID Connect. 3. Create a GitHub secret.

Section	Tasks
Build	1. Set up the environment. 2. Build the app.
Image	1. Create a VM Image. 2. Create a virtual machine.

Create a user-managed identity

You'll need a user-managed identity for Azure Image Builder(AIB) to distribute images. Your Azure user-assigned managed identity will be used during the image build to read and write images to a Shared Image Gallery.

1. Create a user-managed identity with Azure CLI or the Azure portal. Write down the name of your managed identity.

2. Customize this JSON code. Replace the placeholders for `{subscriptionID}` and `{rgName}` with your subscription ID and resource group name.

YAML

```
{
"properties": {
    "roleName": "Image Creation Role",
    "IsCustom": true,
    "description": "Azure Image Builder access to create resources for
the image build",
    "assignableScopes": [
      "/subscriptions/{subscriptionID}/resourceGroups/{rgName}"
    ],
    "permissions": [
        {
            "actions": [
                "Microsoft.Compute/galleries/read",
                "Microsoft.Compute/galleries/images/read",
                "Microsoft.Compute/galleries/images/versions/read",
                "Microsoft.Compute/galleries/images/versions/write",
                "Microsoft.Compute/images/write",
                "Microsoft.Compute/images/read",
                "Microsoft.Compute/images/delete"
            ],
            "notActions": [],
            "dataActions": [],
            "notDataActions": []
        }
    ]
} } ```
```

3. Use this JSON code to create a new custom role with JSON.

Generate deployment credentials

Service principal

Create a service principal with the az ad sp create-for-rbac command in the Azure CLI. Run this command with Azure Cloud Shell in the Azure portal or by selecting the **Try it** button.

Azure CLI

```
az ad sp create-for-rbac --name "myML" --role contributor \
                            --scopes /subscriptions/<subscription-
id>/resourceGroups/<group-name> \
                            --sdk-auth
```

In the example above, replace the placeholders with your subscription ID, resource group name, and app name. The output is a JSON object with the role assignment credentials that provide access to your App Service app similar to below. Copy this JSON object for later.

Output

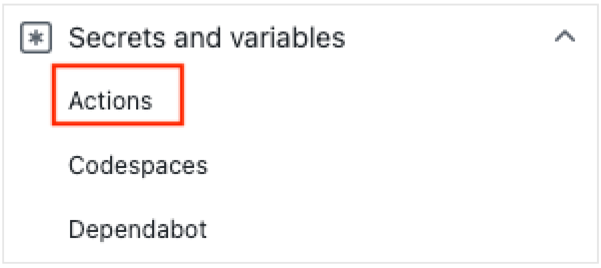
```
  {
    "clientId": "<GUID>",
    "clientSecret": "<GUID>",
    "subscriptionId": "<GUID>",
    "tenantId": "<GUID>",
    (...)
  }
```

Create GitHub secrets

Service principal

1. In GitHub, go to your repository.

2. Select **Security** > **Secrets and variables** > **Actions**.



3. Select **New repository secret**.

4. Paste the entire JSON output from the Azure CLI command into the secret's value field. Give the secret the name `AZURE_CREDENTIALS`.

5. Select **Add secret**.

Use the Azure login action

Use your GitHub secret with the Azure Login action to authenticate to Azure.

Service principal

In this workflow, you authenticate using the Azure login action with the service principal details stored in `secrets.AZURE_CREDENTIALS`. Then, you run an Azure CLI action. For more information about referencing GitHub secrets in a workflow file, see Using encrypted secrets in a workflow in GitHub Docs.

YAML

```yaml
on: [push]

name: Create Custom VM Image

jobs:
  build-image:
    runs-on: ubuntu-latest
    steps:
      - name: Log in with Azure
        uses: azure/login@v1
        with:
          creds: '${{ secrets.AZURE_CREDENTIALS }}'
```

Configure Java

Set up the Java environment with the Java Setup SDK action . For this example, you'll set up the environment, build with Maven, and then output an artifact.

GitHub artifacts are a way to share files in a workflow between jobs. You'll create an artifact to hold the JAR file and then add it to the virtual machine image.

Service principal

YAML

```yaml
on: [push]

name: Create Custom VM Image

jobs:
  build-image:
    runs-on: ubuntu-latest
    steps:
    - name: Checkout
      uses: actions/checkout@v2

    - name: Login via Az module
      uses: azure/login@v1
      with:
        creds: ${{secrets.AZURE_CREDENTIALS}}

    - name: Setup Java 1.8.x
      uses: actions/setup-java@v1
      with:
        java-version: '1.8.x'

    - name: Build Java
      run: mvn --batch-mode --update-snapshots verify

    - run: mkdir staging && cp target/*.jar staging
    - uses: actions/upload-artifact@v2
      with:
        name: Package
        path: staging
```

Build your image

Use the Build Azure Virtual Machine Image action to create a custom virtual machine image.

Replace the placeholders for `{subscriptionID}`, `{rgName}` and `{Identity}` with your subscription ID, resource group name, and managed identity name. Replace the values of `{galleryName}` and `{imageName}` with your image gallery name and your image name.

```yaml
    - name: Create App Baked Image
      id: imageBuilder
      uses: azure/build-vm-image@v0
      with:
        location: 'eastus2'
        resource-group-name: '{rgName}'
        managed-identity: '{Identity}' # Managed identity
        source-os-type: 'windows'
        source-image-type: 'platformImage'
        source-image: MicrosoftWindowsServer:WindowsServer:2019-
Datacenter:latest #unique identifier of source image
        dist-type: 'SharedImageGallery'
        dist-resource-id:
'/subscriptions/{subscriptionID}/resourceGroups/{rgName}/providers/Microsoft
.Compute/galleries/{galleryName}/images/{imageName}/versions/0.1.${{
GITHUB.RUN_ID }}' #Replace with the resource id of your shared image
gallery's image definition
        dist-location: 'eastus2'
```

Virtual Machine action arguments

Input	Required	Description
`resource-group-name`	Yes	The resource group used for storage and saving artifacts during the build process.
`image-builder-template-name`	No	The name of the image builder template resource used.
`location`	Yes	The location where Azure Image Builder will run. See supported locations.
`build-timeout-in-minutes`	No	Time after which the build is canceled. Defaults to 240.
`vm-size`	Optional	By default, `Standard_D1_v2` will be used. See virtual machine sizes.
`managed-identity`	Yes	The user-managed identity you created earlier. Use the full identifier if your identity is in a different resources group. Use the name if it is in the same resource group.
`source-os`	Yes	The OS type of the base image (Linux or Windows)

Input	Required	Description
`source-image-type`	Yes	The base image type that will be used for creating the custom image.
`source-image`	Yes	The resource identifier for base image. A source image should be present in the same Azure region set in the input value of location.
`customizer-source`	No	The directory where you can keep all the artifacts that need to be added to the base image for customization. By default, the value is `${{ GITHUB.WORKSPACE }}/workflow-artifacts.`
`customizer-destination`	No	This is the directory in the customized image where artifacts are copied to.
`customizer-windows-update`	No	For Windows only. Boolean value. If `true`, the image builder will run Windows update at the end of the customizations.
`dist-location`	No	For SharedImageGallery, this is the `dist-type`.
`dist-image-tags`	No	These are user-defined tags that are added to the custom image created (example: `version:beta`).

Create your virtual machine

As a last step, create a virtual machine from your image.

1. Replace the placeholders for `{rgName}`with your resource group name.

2. Add a GitHub secret with the virtual machine password (`VM_PWD`). Be sure to write down the password because you will not be able to see it again. The username is `myuser`.

YAML

```yaml
    - name: CREATE VM
      uses: azure/CLI@v1
      with:
        azcliversion: 2.0.72
        inlineScript: |
        az vm create --resource-group ghactions-vMimage  --name "app-vm-${{
GITHUB.RUN_NUMBER }}"  --admin-username myuser --admin-password "${{
secrets.VM_PWD }}" --location  eastus2 \
            --image "${{ steps.imageBuilder.outputs.custom-image-uri }}"
```

Complete YAML

Service principal

YAML

```
  on: [push]

  name: Create Custom VM Image

  jobs:
    build-image:
      runs-on: ubuntu-latest
      steps:
      - name: Checkout
        uses: actions/checkout@v2

      - name: Login via Az module
        uses: azure/login@v1
        with:
          creds: ${{secrets.AZURE_CREDENTIALS}}

      - name: Setup Java 1.8.x
        uses: actions/setup-java@v1
        with:
          java-version: '1.8.x'

      - name: Build Java
        run: mvn --batch-mode --update-snapshots verify

      - run: mkdir staging && cp target/*.jar staging
      - uses: actions/upload-artifact@v2
        with:
          name: Package
          path: staging

      - name: Create App Baked Image
        id: imageBuilder
        uses: azure/build-vm-image@v0
        with:
          location: 'eastus2'
          resource-group-name: '{rgName}'
          managed-identity: '{Identity}' # Managed identity
          source-os-type: 'windows'
          source-image-type: 'platformImage'
          source-image: MicrosoftWindowsServer:WindowsServer:2019-
Datacenter:latest #unique identifier of source image
          dist-type: 'SharedImageGallery'
          dist-resource-id:
'/subscriptions/{subscriptionID}/resourceGroups/{rgName}/providers/Micro
soft.Compute/galleries/{galleryName}/images/{imageName}/versions/0.1.${{
GITHUB.RUN_ID }}' #Replace with the resource id of your shared image
gallery's image definition
```

```
            dist-location: 'eastus2'

        - name: CREATE VM
          uses: azure/CLI@v1
          with:
            azcliversion: 2.0.72
            inlineScript: |
            az vm create --resource-group ghactions-vMimage  --name "app-
vm-${{ GITHUB.RUN_NUMBER }}"  --admin-username myuser --admin-password
"${{ secrets.VM_PWD }}" --location  eastus2 \
                --image "${{ steps.imageBuilder.outputs.custom-image-uri
}}"
```

Next steps

- Learn how to deploy to Azure.

Work with Azure DevOps and GitHub

Article • 09/22/2022 • 2 minutes to read

Review the following article to learn how Azure DevOps works with GitHub.

- Connect Azure Boards with GitHub
- Link Azure Boards work items to GitHub commits, pull requests and issues
- Use Azure Pipelines to build GitHub repositories
- Create a GitHub Release from Azure Pipelines

Use Visual Studio or Visual Studio Code to deploy apps from GitHub

Article • 09/22/2022 • 2 minutes to read

Review the following article to learn how to use Visual Studio or Visual Studio Code with GitHub.

- Deploy to Azure App Service using Visual Studio Code
- Visual Studio subscription with GitHub offer

And, the following Marketplace extensions provide developer support for integrating with GitHub.

- GitHub extension for Visual Studio
- GitHub extension for Visual Studio Code

You can also use Visual Studio and Visual Studio Code to create your own actions.

- Tutorial: Create a GitHub Action with .NET