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

Azure Developer CLI (azd)

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 • 03/13/2024

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

Microsoft developed GitHub Actions for Azure and designed them 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 can use 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. When you use a service principal you can use OpenID Connect or a secret.

The Azure App Service action supports using a publish profile or service principal. See Application and service principal objects in Microsoft Entra ID 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. You can also use Azure login with community-contributed actions like Enhanced Azure key vault that aren't officially supported by Microsoft.

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.

- Azure Spring Cloud
- 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

Use GitHub Actions to connect to Azure

Article • 08/08/2024

Learn how to use Azure Login action with either Azure PowerShell action or Azure CLI action 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 action.

The Azure Login action supports different ways of authenticating with Azure:

- Sign in with OpenID Connect using a Microsoft Entra application or a user-assigned managed identity
- Sign in with a managed identity configured on an Azure virtual machine (Only available for self-hosted GitHub runners)
- Sign in with a service principal and secret (Not recommended)

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

You can also use the Azure Login action to connect to public or sovereign clouds including Azure Government and Azure Stack Hub.

Connect with other Azure services

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

Expand table

Service	Tutorial
Microsoft Entra ID	Sign in to GitHub Enterprise with Microsoft Entra ID (single sign-on)
Power BI	Connect Power BI with GitHub
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

Use the Azure Login action with OpenID Connect

Article • 08/08/2024

Learn how to securely authenticate to Azure services from GitHub Actions workflows using Azure Login action with OpenID Connect (OIDC).

In this tutorial, you learn how to:

✓ Create GitHub secrets for the credentials of a Microsoft Entra application/user-assigned managed identity
✓ Set up Azure Login with OpenID Connect authentication in GitHub Actions workflows

Prerequisites

To use Azure Login action with OIDC, you need to configure a federated identity credential on a Microsoft Entra application or a user-assigned managed identity.

Option 1: Microsoft Entra application

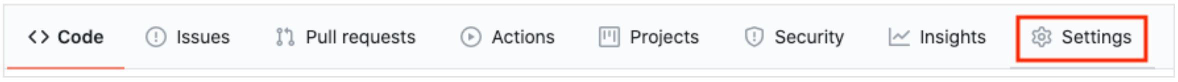
- Create a Microsoft Entra application with a service principal by Azure portal, Azure CLI, or Azure PowerShell.
- Copy the values for **Client ID**, **Subscription ID**, and **Directory (tenant) ID** to use later in your GitHub Actions workflow.
- Assign an appropriate role to your service principal by Azure portal, Azure CLI, or Azure PowerShell.
- Configure a federated identity credential on a Microsoft Entra application to trust tokens issued by GitHub Actions to your GitHub repository.

Option 2: User-assigned managed identity

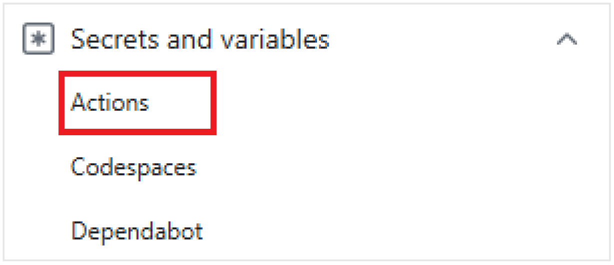
- Create a user-assigned managed identity.
- Copy the values for **Client ID**, **Subscription ID**, and **Directory (tenant) ID** to use later in your GitHub Actions workflow.
- Assign an appropriate role to your user-assigned managed identity.
- Configure a federated identity credential on a user-assigned managed identity to trust tokens issued by GitHub Actions to your GitHub repository.

Create GitHub secrets

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



2. Select **Security > Secrets and variables > Actions > New repository secret**.



> ⓘ **Note**
>
> To enhance workflow security in public repositories, use environment secrets instead of repository secrets. If the environment requires approval, a job cannot access environment secrets until one of the required reviewers approves it.

3. Create secrets for `AZURE_CLIENT_ID`, `AZURE_TENANT_ID`, and `AZURE_SUBSCRIPTION_ID`. Copy these values from your Microsoft Entra application or user-assigned managed identity for your GitHub secrets:

Expand table

GitHub secret	Microsoft Entra application or user-assigned managed identity
AZURE_CLIENT_ID	Client ID
AZURE_SUBSCRIPTION_ID	Subscription ID
AZURE_TENANT_ID	Directory (tenant) ID

> ⓘ **Note**
>
> For security reasons, we recommend using GitHub Secrets rather than passing values directly to the workflow.

Set up Azure Login action with OpenID Connect in GitHub Actions workflows

Your GitHub Actions workflow uses OpenID Connect to authenticate with Azure. Once you have a working Azure Login step, you can use the Azure PowerShell action or Azure CLI action. You can also use other Azure actions, like Azure webapp deploy and Azure functions.

To learn more about this interaction, see the GitHub Actions documentation.

In this example, you use OpenID Connect to authenticate with Azure with the Azure login action. The example uses GitHub secrets stored before for the `client-id`, `tenant-id`, and `subscription-id` values.

The Azure Login action includes an optional `audience` input parameter that defaults to `api://AzureADTokenExchange`, available for public clouds. For non-public clouds, update this parameter with the appropriate values. You can also customize this parameter for specific audience values.

The workflow sample to only run Azure CLI

This workflow authenticates with OpenID Connect and uses Azure CLI to get the details of the connected subscription.

YAML

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

jobs:
  test:
    permissions:
      id-token: write # Require write permission to Fetch an OIDC token.

    runs-on: ubuntu-latest
    steps:
    - name: Azure CLI Login
      uses: azure/login@v2
      with:
        client-id: ${{ secrets.AZURE_CLIENT_ID }}
        tenant-id: ${{ secrets.AZURE_TENANT_ID }}
        subscription-id: ${{ secrets.AZURE_SUBSCRIPTION_ID }}

    - name: Azure CLI script
      uses: azure/cli@v2
      with:
        azcliversion: latest
        inlineScript: |
          az account show
          # You can write your Azure CLI inline scripts here.
```

The workflow sample to run both Azure CLI and Azure PowerShell

This workflow authenticates with OpenID Connect and uses both Azure CLI and Azure PowerShell to get the details of the connected subscription.

YAML

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

jobs:
  test:
    permissions:
      id-token: write # Require write permission to Fetch an OIDC token.

    runs-on: ubuntu-latest
    steps:
    - name: Azure Login
      uses: azure/login@v2
      with:
        client-id: ${{ secrets.AZURE_CLIENT_ID }}
        tenant-id: ${{ secrets.AZURE_TENANT_ID }}
        subscription-id: ${{ secrets.AZURE_SUBSCRIPTION_ID }}
        enable-AzPSSession: true

    - name: Azure CLI script
      uses: azure/cli@v2
      with:
        azcliversion: latest
        inlineScript: |
          az account show
          # You can write your Azure CLI inline scripts here.

    - name: Azure PowerShell script
      uses: azure/powershell@v2
      with:
        azPSVersion: latest
        inlineScript: |
          Get-AzContext
          # You can write your Azure PowerShell inline scripts here.
```

Connect to Azure Government clouds and Azure Stack Hub clouds

To log in to one of the Azure Government clouds or Azure Stack, set the parameter `environment` to one of the following supported values: `AzureUSGovernment`, `AzureChinaCloud`, `AzureGermanCloud`, or `AzureStack`. If this parameter isn't specified, it takes the default value `AzureCloud` and connects to the Azure Public Cloud.

```yaml
jobs:
  test:
    permissions:
      id-token: write # Require write permission to Fetch an OIDC token.
    runs-on: ubuntu-latest
    steps:
    - name: Login to Azure US Gov Cloud with both Azure CLI and Azure
Powershell
      uses: azure/login@v2
        with:
          client-id: ${{ secrets.AZURE_CLIENT_ID }}
          tenant-id: ${{ secrets.AZURE_TENANT_ID }}
          subscription-id: ${{ secrets.AZURE_SUBSCRIPTION_ID }}
          environment: 'AzureUSGovernment'
          audience: api://AzureADTokenExchangeUSGov
          enable-AzPSSession: true
```

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

Use the Azure Login action with a managed identity

Article • 08/08/2024

On a virtual machine configured with managed identities in Azure, you can sign in Azure Login action using the managed identity. You don't need to manage credentials, as they aren't accessible to you. There are two types of managed identities for you to choose: **system-assigned managed identities** or **user-assigned managed identities**.

In this tutorial, you learn how to:

- ✓ Create GitHub secrets for system/user-assigned managed identity
- ✓ Set up Azure Login for system/user-assigned managed identity in GitHub Actions workflows

> **Note**
>
> Login with managed identity is only supported for self-hosted runners on Azure virtual machines. For other users, please refer to **Sign in with OpenID Connect** or **Sign in with a service principal and secret**.

Prerequisites

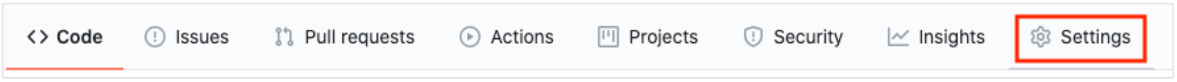
- Create an Azure virtual machine
  - Create a Windows virtual machine
  - Create a Linux virtual machine
- Configure managed identity on the Azure virtual machine
- Install required software on the Azure virtual machine
  - Install Azure CLI
  - install Docker
  - Install PowerShell
  - Install Azure PowerShell
- Configure the Azure virtual machine as a GitHub self-hosted runner

Use the Azure Login action with system-assigned managed identity

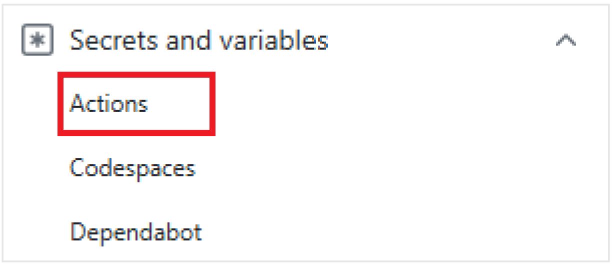
Learn how to securely authenticate to Azure services from GitHub Actions workflows using Azure Login action with system-assigned managed identity that configured on a virtual machine.

Create GitHub secrets for system-assigned managed identity

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



2. Select **Security** > **Secrets and variables** > **Actions** > **New repository secret**.



> **Note**
>
> To enhance workflow security in public repositories, use environment secrets instead of repository secrets. If the environment requires approval, a job cannot access environment secrets until one of the required reviewers approves it.

3. Create secrets for `AZURE_TENANT_ID`, and `AZURE_SUBSCRIPTION_ID`. Copy these values from your user-assigned managed identity for your GitHub secrets:

Expand table

GitHub secret	System-assigned managed identity
AZURE_SUBSCRIPTION_ID	Subscription ID
AZURE_TENANT_ID	Directory (tenant) ID

> **Note**
>
> For security reasons, we recommend using GitHub Secrets rather than passing values directly to the workflow.

Set up Azure Login action with system-assigned managed identity in GitHub Actions workflows

In this example, you use the system-assigned managed identity to authenticate with Azure with the Azure login action. The example uses GitHub secrets for the `subscription-id`, and `tenant-id` values. The self-hosted runner has been labeled `self-hosted` on GitHub.

YAML

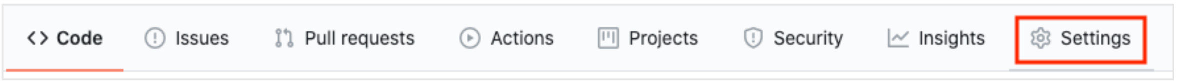
```yaml
name: Run Azure Login with system-assigned managed identity
on: [push]

jobs:
  test:
    runs-on: [self-hosted] # Specify the label of your self-hosted runner
here
    steps:
    - name: Azure login
      uses: azure/login@v2
      with:
        auth-type: IDENTITY
        tenant-id: ${{ secrets.AZURE_TENANT_ID }}
        subscription-id: ${{ secrets.AZURE_SUBSCRIPTION_ID }}
        enable-AzPSSession: true

    # Azure CLI action only supports linux self-hosted runners for now.
    # If you want to execute the Azure CLI script on a windows self-hosted
runner, you can execute it directly in `run`.
    - name: Azure CLI script
      uses: azure/cli@v2
      with:
        azcliversion: latest
        inlineScript: |
          az account show
          # You can write your Azure CLI inline scripts here.

    - name: Azure PowerShell script
      uses: azure/powershell@v2
      with:
        azPSVersion: latest
        inlineScript: |
          Get-AzContext
          # You can write your Azure PowerShell inline scripts here.
```

Use the Azure Login action with user-assigned managed identity

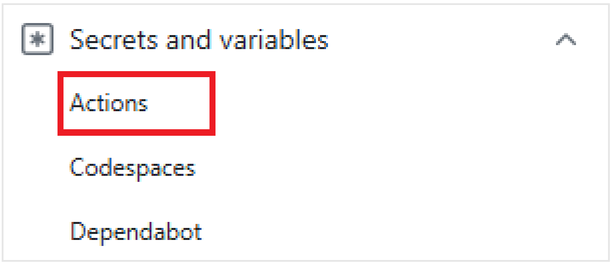
Learn how to securely authenticate to Azure services from GitHub Actions workflows using Azure Login action with user-assigned managed identity that configured on a virtual machine.

Create GitHub secrets for user-assigned managed identity

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



2. Select **Security** > **Secrets and variables** > **Actions** > **New repository secret**.



> Note
>
> To enhance workflow security in public repositories, use environment secrets instead of repository secrets. If the environment requires approval, a job cannot access environment secrets until one of the required reviewers approves it.

3. Create secrets for `AZURE_CLIENT_ID`, `AZURE_TENANT_ID`, and `AZURE_SUBSCRIPTION_ID`. Copy these values from your user-assigned managed identity for your GitHub secrets:

Expand table

GitHub secret	User-assigned managed identity
AZURE_CLIENT_ID	Client ID
AZURE_SUBSCRIPTION_ID	Subscription ID
AZURE_TENANT_ID	Directory (tenant) ID

> Note

For security reasons, we recommend using GitHub Secrets rather than passing values directly to the workflow.

Set up Azure Login action with user-assigned managed identity in GitHub Actions workflows

In this example, you use the user-assigned managed identity to authenticate with Azure with the Azure login action. The example uses GitHub secrets for the `client-id`, `subscription-id`, and `tenant-id` values. The self-hosted runner has been labeled `self-hosted` on GitHub.

YAML

```yaml
name: Run Azure Login with user-assigned managed identity
on: [push]

jobs:
  test:
    runs-on: [self-hosted] # Specify the label of your self-hosted runner
here
    steps:
    - name: Azure login
      uses: azure/login@v2
      with:
        auth-type: IDENTITY
        client-id: ${{ secrets.AZURE_CLIENT_ID }}
        tenant-id: ${{ secrets.AZURE_TENANT_ID }}
        subscription-id: ${{ secrets.AZURE_SUBSCRIPTION_ID }}
        enable-AzPSSession: true

    # Azure CLI action only supports linux self-hosted runners for now.
    # If you want to execute the Azure CLI script on a windows self-hosted
runner, you can execute it directly in `run`.
    - name: Azure CLI script
      uses: azure/cli@v2
      with:
        azcliversion: latest
        inlineScript: |
          az account show
          # You can write your Azure CLI inline scripts here.

    - name: Azure PowerShell script
      uses: azure/powershell@v2
      with:
        azPSVersion: latest
        inlineScript: |
          Get-AzContext
          # You can write your Azure PowerShell inline scripts here.
```

Use the Azure Login action with a client secret

Article • 08/08/2024

Learn how to create a service principal with a client secret and securely authenticate to Azure services from GitHub Actions workflows using Azure Login action.

In this tutorial, you learn how to:

- ✓ Create a GitHub secret for the service principal
- ✓ Set up Azure Login for service principal secret in GitHub Actions workflows

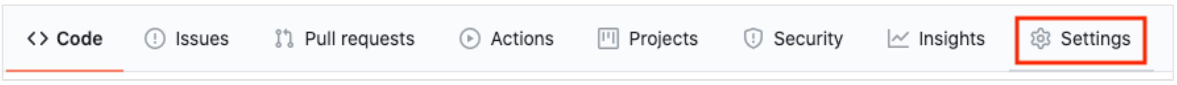
> ⚠ **Warning**
>
> Treat your client secrets with care to prevent leaks. Unauthorized disclosure can compromise security. Store secrets securely and share only with authorized ones.

Prerequisites

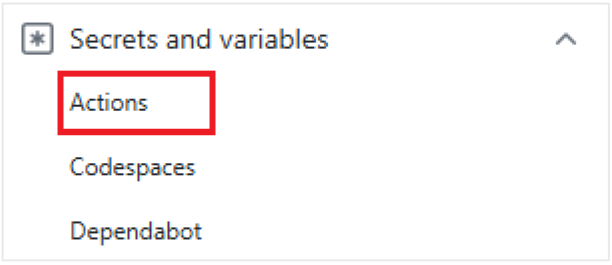
- Create a Microsoft Entra application with a service principal by Azure portal, Azure CLI, or Azure PowerShell.
- Create a client secret for your service principal by Azure portal, Azure CLI, or Azure PowerShell.
- Copy the values for **Client ID**, **Client Secret**, **Subscription ID**, and **Directory (tenant) ID** to use later in your GitHub Actions workflow.
- Assign an appropriate role to your service principal by Azure portal, Azure CLI, or Azure PowerShell.

Create a GitHub secret for the service principal

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



2. Select **Security** > **Secrets and variables** > **Actions** > **New repository secret**.



> **Note**
>
> To enhance workflow security in public repositories, use [environment secrets](#) instead of repository secrets. If the environment requires approval, a job cannot access environment secrets until one of the required reviewers approves it.

3. Create a GitHub Actions secret `AZURE_CREDENTIALS` in the following format. Copy these values from your service principal.

JSON

```json
{
    "clientId": "<Client ID>",
    "clientSecret": "<Client Secret>",
    "subscriptionId": "<Subscription ID>",
    "tenantId": "<Tenant ID>"
}
```

Expand table

GitHub secret	Service principal
clientId	Client ID
clientSecret	Client Secret
subscriptionId	Subscription ID
tenantId	Directory (tenant) ID

Set up Azure Login action with the Service Principal secret in GitHub Actions workflows

To authenticate to Azure in GitHub Actions workflows using the service principal secret, you need to use the [Azure Login action](#).

Use the Azure Login action with both Azure CLI action and Azure PowerShell action

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

YAML

```yaml
name: Run Azure Login with the Service Principal secret
on: [push]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
    - name: Azure Login action
      uses: azure/login@v2
      with:
        creds: ${{ secrets.AZURE_CREDENTIALS }}
        enable-AzPSSession: true

    - name: Azure CLI script
      uses: azure/cli@v2
      with:
        azcliversion: latest
        inlineScript: |
          az group show --name "<YOUR RESOURCE GROUP>"
          # You can write your Azure CLI inline scripts here.

    - name: Azure PowerShell action
      uses: azure/powershell@v2
      with:
        azPSVersion: latest
        inlineScript: |
          Get-AzResourceGroup -Name "<YOUR RESOURCE GROUP>"
          # You can write your Azure PowerShell inline scripts here.
```

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

Integrate Azure Key Vault into a GitHub Actions workflow

07/01/2025

Integrate Azure Key Vault into your GitHub Actions workflow to securely manage sensitive credentials in one place. This approach reduces the risk of accidental exposure or unauthorized access to sensitive data.

This GitHub Actions sample workflow demonstrates how to securely retrieve secrets from Azure Key Vault using OpenID Connect (OIDC) authentication.

Prerequisites

- Configure a federated identity credential on a Microsoft Entra application or a user-assigned managed identity. Learn how in Authenticate to Azure from GitHub Actions by OpenID Connect. When you set up your federated credential, store these secrets in GitHub:
  - `AZURE_CLIENT_ID`: Your Azure service principal's client ID.
  - `AZURE_TENANT_ID`: Your Azure AD tenant ID.
  - `AZURE_SUBSCRIPTION_ID`: Your Azure subscription ID.
  - `KEYVAULT_NAME`: Your Key Vault name.
- Grant permissions: Make sure the service principal has appropriate access to the Key Vault (example, "Key Vault Secrets User" role).
- Replace `<SECRET_NAME>` with your Key Vault secret name.

GitHub Actions workflow sample

What the workflow does:

- Triggers on pushes to the main branch
- Uses OIDC authentication to connect to Azure (no passwords stored in GitHub)
- Retrieves a secret from Azure Key Vault
- Masks the secret value with `::add-mask::` to prevent it from appearing in logs
- Makes the secret available as an environment variable for subsequent steps

YAML

```yaml
name: Access Azure Key Vault and pass secret to workflow

on:
  push:
    branches:
      - main

permissions:
  id-token: write
  contents: read

jobs:
  get-secret:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Azure Login
        uses: azure/login@v1
        with:
          client-id: ${{ secrets.AZURE_CLIENT_ID }}
          tenant-id: ${{ secrets.AZURE_TENANT_ID }}
          subscription-id: ${{ secrets.AZURE_SUBSCRIPTION_ID }}

      - name: Retrieve secret from Key Vault
        id: keyvault
        uses: azure/CLI@v1
        with:
          inlineScript: |
            SECRET_VALUE=$(az keyvault secret show --name <SECRET_NAME> --vault-
name ${{ secrets.KEYVAULT_NAME }} --query value -o tsv)
            echo "::add-mask::$SECRET_VALUE"
            echo "SECRET_VALUE=$SECRET_VALUE" >> $GITHUB_ENV
      - name: Use retrieved secret
        run: echo "The secret is successfully retrieved!"

      - name: Use SECRET_VALUE in deployment
        run: |
          ./deploy.sh
        env:
          SECRET_VALUE: ${{ env.SECRET_VALUE }}
```

Additional resources

- Authenticate to Azure from GitHub Actions by OpenID Connect
- About Azure Key Vault

Deploy apps from GitHub to Azure

Article • 07/07/2023

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 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 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 • 12/20/2023

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.

For Azure SQL Managed Instance, use Azure CLI and the Azure CLI action .

- 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

Manage Azure Policies with GitHub

Article • 09/22/2022

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 • 02/12/2025

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.
- An Azure Compute Gallery with an image.
  - Create an Azure Compute Gallery.
  - Create an image.

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:

Expand table

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

Section	Tasks
	3. Create a GitHub secret.
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.

4. In Azure portal, open your Azure Compute Gallery and go to **Access control (IAM)**.

5. Select **Add role assignment** and assign the Image Creation Role to your user-managed identity.

Generate deployment credentials

OpenID Connect

To use Azure Login action with OIDC, you need to configure a federated identity credential on a Microsoft Entra application or a user-assigned managed identity.

Option 1: Microsoft Entra application

- Create a Microsoft Entra application with a service principal by Azure portal, Azure CLI, or Azure PowerShell.
- Copy the values for **Client ID**, **Subscription ID**, and **Directory (tenant) ID** to use later in your GitHub Actions workflow.
- Assign an appropriate role to your service principal by Azure portal, Azure CLI, or Azure PowerShell.
- Configure a federated identity credential on a Microsoft Entra application to trust tokens issued by GitHub Actions to your GitHub repository.

Option 2: User-assigned managed identity

- Create a user-assigned managed identity.
- Copy the values for **Client ID**, **Subscription ID**, and **Directory (tenant) ID** to use later in your GitHub Actions workflow.
- Assign an appropriate role to your user-assigned managed identity.
- Configure a federated identity credential on a user-assigned managed identity to trust tokens issued by GitHub Actions to your GitHub repository.

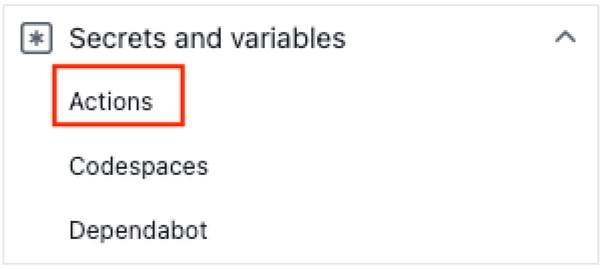
Create GitHub secrets

OpenID Connect

You need to provide your application's **Client ID**, **Directory (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. In GitHub, go to your repository.

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



3. Select **New repository secret**.

> Note
>
> To enhance workflow security in public repositories, use **environment secrets** instead of repository secrets. If the environment requires approval, a job cannot access environment secrets until one of the required reviewers approves it.

4. Create secrets for `AZURE_CLIENT_ID`, `AZURE_TENANT_ID`, and `AZURE_SUBSCRIPTION_ID`. Copy these values from your Microsoft Entra application or user-assigned managed identity for your GitHub secrets:

Expand table

GitHub secret	**Microsoft Entra application or user-assigned managed identity**
AZURE_CLIENT_ID	Client ID
AZURE_SUBSCRIPTION_ID	Subscription ID
AZURE_TENANT_ID	Directory (tenant) ID

> Note

For security reasons, we recommend using GitHub Secrets rather than passing values directly to the workflow.

Use the Azure login action

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

OpenID Connect

For Open ID Connect you'll use a federated credential associated with your Active Directory app.

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@v2
        with:
          client-id: ${{ secrets.AZURE_CLIENT_ID }}
          tenant-id: ${{ secrets.AZURE_TENANT_ID }}
          subscription-id: ${{ secrets.AZURE_SUBSCRIPTION_ID }}
```

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.

OpenID Connect

YAML

```yaml
on: [push]

name: Create Custom VM Image

jobs:
  build-image:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        java: [ '17' ]

    steps:
    - name: Checkout
      uses: actions/checkout@v3

    - name: Login via Az module
      uses: azure/login@v2
      with:
        creds: ${{secrets.AZURE_CREDENTIALS}}

    - name: Set up JDK ${{matrix.java}}
      uses: actions/setup-java@v2
      with:
        java-version: ${{matrix.java}}
        distribution: 'adopt'
        cache: maven
    - name: Build with Maven Wrapper
      run: ./mvnw -B package

    - 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.

ⓘ Note

If the Create App Baked Image action fails with a permission error, verify that you have assigned the Image Creation Role to your user-managed identity.

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

Expand table

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.

Input	Required	Description
`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)
`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

OpenID Connect

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@v2
      with:
        client-id: ${{ secrets.AZURE_CLIENT_ID }}
        tenant-id: ${{ secrets.AZURE_TENANT_ID }}
        subscription-id: ${{ secrets.AZURE_SUBSCRIPTION_ID }}

    - 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

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

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