Azure Artifacts

1. Introduction to Azure Artifacts

What is Azure Artifacts?

Azure Artifacts is a fully integrated package management solution in Azure DevOps. It allows you to create and share packages across teams and projects. Azure Artifacts is designed to work with popular package management systems, making it easy to manage dependencies and distribute your own packages.

Key Features and Benefits:

- **Centralized Package Management:** Host and share packages from a central location within your organization.
- **Support for Multiple Package Types:** Manage NuGet, npm, Maven, and Python packages all in one place.
- **Seamless Integration:** Integrates with Azure Pipelines, making it easy to include package management in your CI/CD workflows.
- **Security and Compliance:** Control who has access to your packages with fine-grained permissions and audit trails.
- **Scalability:** Designed to handle the needs of large organizations with multiple teams and projects.

Supported Package Types

Azure Artifacts supports a wide range of package types, making it a versatile tool for managing dependencies across various technologies:

- **NuGet:** For .NET applications.
- **npm:** For Node.js applications.
- **Maven:** For Java applications.
- **Python:** For Python applications.
- **Universal Packages:** For any type of file or binary not natively supported by the other package managers.

2. Setting Up Azure Artifacts

Creating an Azure DevOps Project

Before using Azure Artifacts, you need to create a project in Azure DevOps. This project will house your artifacts, feeds, and other related resources.

1. **Create a New Project:**
   - Go to the Azure DevOps portal.
   - Click on "New Project."
   - Enter a project name, description, and select the appropriate visibility (public or private).
   - Click "Create" to initialize the project.
2. **Accessing Azure Artifacts:**
   - Once your project is set up, navigate to the "Artifacts" tab in Azure DevOps.
   - This is where you will create and manage your feeds.

Setting Up Your First Feed

Feeds in Azure Artifacts are used to organize and store your packages. A feed can contain multiple packages and can be configured to be public or private.

1. **Create a New Feed:**
   - In the "Artifacts" tab, click on "New Feed."
   - Enter a name for your feed and select its visibility (e.g., public or private).
   - Choose whether to include upstream sources, which allow you to pull packages from external repositories like npmjs.com or NuGet.org.
   - Click "Create" to set up the feed.
2. **Configuring Feed Settings:**
   - After creating the feed, you can configure additional settings such as permissions, retention policies, and upstream sources.
   - Navigate to "Feed settings" to adjust these configurations according to your project’s needs.

3. Managing Packages

Publishing Packages to Azure Artifacts

Publishing packages to Azure Artifacts is straightforward and can be done through the command line, integrated development environments (IDEs), or automated pipelines.

1. **Publishing a NuGet Package:**
   - Open a terminal in your project directory.
   - Use the dotnet CLI to create a NuGet package:

```
dotnet pack -c Release
```

   - Push the package to Azure Artifacts using the nuget CLI or directly through Visual Studio:

```
dotnet nuget push *.nupkg -s <feed-url> --api-key <your-api-key>
```

2. **Publishing an npm Package:**
   - Navigate to your Node.js project directory.
   - Create a package using the npm CLI:

```
npm pack
```

   - Publish the package to your Azure Artifacts feed:

```
npm publish --registry <feed-url>
```

3. **Publishing Maven and Python Packages:**
   - Similar commands and configurations apply for Maven and Python packages using tools like mvn and twine.

Consuming Packages from Azure Artifacts

To use packages hosted in Azure Artifacts, you need to configure your package manager to point to the appropriate feed.

1. **Consuming NuGet Packages:**
   - Add your Azure Artifacts feed as a package source in Visual Studio or via the NuGet CLI:

```
nuget sources add -Name "MyFeed" -Source "<feed-url>"
```

   - Install the package using the NuGet CLI or Visual Studio:

```
dotnet add package <package-name> --source <feed-url>
```

2. **Consuming npm Packages:**
    - Update your .npmrc file to include the feed URL:

```
registry=<feed-url>
```

    - Install the package using npm:

```
npm install <package-name>
```

3. **Consuming Maven and Python Packages:**
    - Update the respective configuration files (pom.xml, requirements.txt) to include the feed URLs for Maven and Python packages.

Promoting and Deleting Packages

Azure Artifacts allows you to promote packages between different views (e.g., Development, Staging, Production) and delete packages when they are no longer needed.

1. **Promoting a Package:**
    - In the Azure Artifacts UI, navigate to the package you want to promote.
    - Select the "Promote" option and choose the target view (e.g., from Development to Production).
    - This makes the package available in the selected view without affecting other views.
2. **Deleting a Package:**
    - To delete a package, navigate to the package details page in Azure Artifacts.
    - Select "Delete" and confirm the deletion.
    - Deleted packages can be restored within a specified retention period, after which they are permanently removed.

4. Working with Feeds

Public vs. Private Feeds

Azure Artifacts allows you to create both public and private feeds, depending on your collaboration needs and security requirements.

1. **Public Feeds:**
    - Public feeds can be accessed by anyone with the feed URL, making them ideal for sharing open-source packages.
    - Be cautious about exposing sensitive packages in public feeds.
2. **Private Feeds:**
    - Private feeds are restricted to specific users or groups within your Azure DevOps organization.

  - Use private feeds for internal packages, ensuring that only authorized team members have access.

Setting Up Permissions

Permissions in Azure Artifacts control who can view, publish, and manage packages in your feeds.

1. **Configuring Feed Permissions:**
   - Navigate to the "Feed settings" and select "Permissions."
   - Add users or groups and assign roles such as "Reader," "Contributor," or "Owner."
   - Fine-tune permissions to control access to specific packages or views within a feed.
2. **Using Azure DevOps Security Groups:**
   - Leverage Azure DevOps security groups to manage permissions for large teams.
   - Assign feed permissions to these groups to simplify administration.

Connecting Feeds to External Sources

Upstream sources in Azure Artifacts allow you to connect your feed to external repositories, enabling seamless access to external packages.

1. **Adding Upstream Sources:**
   - In the "Feed settings," navigate to "Upstream sources."
   - Add external sources like npmjs.com, NuGet.org, or your own private feeds.
   - This allows you to cache and consume packages from external repositories within your Azure Artifacts feed.
2. **Benefits of Using Upstream Sources:**
   - **Caching:** Upstream sources cache packages from external repositories, reducing download times and ensuring availability even if the external source goes offline.
   - **Unified Package Management:** By consolidating internal and external packages into a single feed, you simplify dependency management and improve package discovery within your organization.
3. **Example Configuration:**
   - For an npm feed, add npmjs.com as an upstream source. This configuration ensures that if a package is not found in your private feed, Azure Artifacts will search npmjs.com:

```
upstreamSources:
 - name: npmjs
   location: https://registry.npmjs.org/
```

5. Integrating Azure Artifacts with CI/CD

Using Azure Artifacts in Build Pipelines

Integrating Azure Artifacts into your CI/CD pipelines allows you to automate the publishing and consuming of packages as part of your build and deployment processes.

1. **Publishing Packages as Part of a Build:**
   - In your Azure Pipelines YAML file, include tasks that package your application and publish the package to Azure Artifacts.
   - Example for a .NET application:

```
steps:
 - task: DotNetCoreCLI@2
   inputs:
     command: 'pack'
     projects: '**/*.csproj'
 - task: NuGetCommand@2
   inputs:
     command: 'push'
     packagesToPush: '**/*.nupkg'
     nuGetFeedType: 'internal'
     publishVstsFeed: '<feed-name>'
```

2. **Consuming Packages During Builds:**
   - Configure your build pipeline to restore dependencies from Azure Artifacts.
   - Example for an npm project:

```
steps:
 - task: Npm@1
   inputs:
     command: 'install'
     workingDir: '$(Build.SourcesDirectory)'
```

3. **Setting Up Pipeline Triggers:**
   - Use triggers in Azure Pipelines to automate package publishing and consuming. For example, you can trigger a pipeline whenever a new package version is published:

```
trigger:
 branches:
   include:
     - main
 paths:
   include:
     - '**/*.nupkg'
```

Integrating with Release Pipelines

Release pipelines can also integrate with Azure Artifacts to automate the deployment of applications that depend on packages hosted in your feeds.

1. **Deploying Applications with Package Dependencies:**
   - In your release pipeline, add steps to install or restore packages from Azure Artifacts before deploying your application.
   - Example:

```
stages:
  - stage: Deploy
    jobs:
      - job: InstallPackages
        steps:
          - script: |
              dotnet restore --source "<feed-url>"
              dotnet publish -c Release
            displayName: 'Restore and Publish'
      - job: DeployApp
        steps:
          - script: |
              az webapp deploy --name <app-name> --resource-group <resource-group> --src-path
$(System.DefaultWorkingDirectory)/**/*.zip
            displayName: 'Deploy to Azure App Service'
```

2. **Automating Package Promotion:**
   - You can automate the promotion of packages across different environments (e.g., from Development to Production) within your release pipeline.
   - Use the Azure Artifacts REST API or CLI to promote packages programmatically as part of your deployment process.

6. Advanced Features

Upstream Sources and Package Management

Upstream sources in Azure Artifacts are essential for managing package dependencies, especially when working with both internal and external packages.

1. **Using Upstream Sources for Dependency Resolution:**
   - When a package is not found in your Azure Artifacts feed, the upstream source is queried. This setup ensures that you can seamlessly integrate third-party packages without duplicating them in your feed.
2. **Caching and Performance Benefits:**
   - Packages fetched from upstream sources are cached in Azure Artifacts, reducing download times for subsequent builds and ensuring high availability even if the external source is temporarily down.

3. **Configuring Upstream Sources in Feeds:**
   - Example configuration for a Maven feed:

```
upstreamSources:
  - name: Maven Central
    location: https://repo.maven.apache.org/maven2/
```

Retention Policies and Clean-Up

Retention policies in Azure Artifacts help manage the storage and lifecycle of packages, preventing unnecessary storage costs and keeping your feeds organized.

1. **Setting Retention Policies:**
   - Define how long packages should be retained before being automatically deleted.
   - Navigate to "Feed settings" > "Retention policies" to configure these settings.
2. **Automating Clean-Up:**
   - Retention policies can automatically delete older versions of packages, keeping only the latest or most frequently used versions.
   - Example:
     - Keep the last 10 versions of each package.
     - Automatically delete packages not used in the last 180 days.
3. **Manual Clean-Up:**
   - If necessary, you can manually delete specific packages or versions via the Azure Artifacts UI.

Package Versioning and Release Management

Proper versioning and release management practices are crucial for maintaining package stability and ensuring that your applications are always using the correct dependencies.

1. **Semantic Versioning:**
   - Follow semantic versioning (SemVer) practices to version your packages, using the format MAJOR.MINOR.PATCH.
   - Example: 1.0.0 for the initial release, 1.1.0 for a backward-compatible feature, and 1.1.1 for a bug fix.
2. **Version Ranges and Constraints:**
   - When consuming packages, use version ranges or constraints to ensure compatibility and avoid breaking changes.
   - Example for npm: ^1.0.0 indicates compatibility with all minor and patch versions within the 1.x.x range.

3. **Managing Pre-releases and Stable Releases:**
    - Use pre-release tags (e.g., 1.0.0-beta) to publish packages that are not yet ready for production.
    - Promote packages from pre-release to stable once they have been thoroughly tested.

7. Hands-on Exercises

Exercise 1: Creating and Publishing a NuGet Package

1. **Objective:**
    - Create a simple NuGet package and publish it to Azure Artifacts.
2. **Steps:**
    - Set up a .NET class library project.
    - Use the dotnet pack command to create a NuGet package.
    - Publish the package to an Azure Artifacts feed using the dotnet nuget push command.
3. **Expected Outcome:**
    - A NuGet package is successfully created and published to your Azure Artifacts feed.

Exercise 2: Setting Up an npm Feed and Publishing a Package

1. **Objective:**
    - Set up an npm feed in Azure Artifacts and publish an npm package.
2. **Steps:**
    - Create a simple Node.js project with a package.json file.
    - Set up an npm feed in Azure Artifacts.
    - Use the npm publish command to publish your package to the feed.
3. **Expected Outcome:**
    - An npm package is successfully published to your Azure Artifacts feed and can be consumed by other projects.

Exercise 3: Integrating Azure Artifacts with a Build Pipeline

1. **Objective:**
   - Integrate Azure Artifacts into a CI pipeline to publish and consume packages.
2. **Steps:**
   - Create a build pipeline in Azure Pipelines.
   - Add tasks to the pipeline to publish a package to Azure Artifacts.
   - Configure the pipeline to consume packages from Azure Artifacts.
3. **Expected Outcome:**
   - The pipeline successfully builds, publishes, and consumes packages using Azure Artifacts.

Exercise 4: Automating Package Promotion and Deployment

1. **Objective:**
   - Automate the promotion of packages from a development environment to production as part of a release pipeline.
2. **Steps:**
   - Set up a multi-stage release pipeline in Azure Pipelines.
   - Use the Azure Artifacts REST API to promote packages between feeds or views.
   - Deploy the promoted package to a production environment.
3. **Expected Outcome:**
   - Packages are automatically promoted and deployed to production, streamlining the release process.

8. Best Practices for Using Azure Artifacts

Organizing Feeds and Packages

1. **Feed Structure:**
   - Organize feeds by project, team, or environment (e.g., Development, Staging, Production) to maintain clarity and avoid confusion.
2. **Package Naming Conventions:**
   - Use consistent naming conventions for packages to make them easily identifiable and searchable.
   - Include relevant information in the package name, such as the project or module name.

3. **View Management:**
    - Use views within feeds to separate packages based on their maturity level (e.g., pre-release, stable).
    - This practice helps teams quickly identify the most appropriate package versions for their needs.

Managing Package Versions and Dependencies

1. **Versioning Strategy:**
    - Implement a clear versioning strategy (e.g., Semantic Versioning) to manage package versions consistently across projects.
2. **Dependency Management:**
    - Regularly review and update dependencies to avoid security vulnerabilities and ensure compatibility with the latest package versions.
3. **Automated Dependency Updates:**
    - Use tools like Dependabot or Renovate to automate dependency updates and keep packages up-to-date with minimal manual intervention.

Securing Feeds and Packages

1. **Access Control:**
    - Restrict access to feeds and packages using Azure DevOps security groups and role-based access control (RBAC).
2. **Audit and Monitoring:**
    - Regularly monitor feed activity and package usage using Azure DevOps audit logs.
    - Implement monitoring tools to detect and alert on unusual activity, such as unauthorized package downloads or uploads.
3. **Secure Package Handling:**
    - Ensure that all packages are scanned for vulnerabilities using tools like SonarQube or WhiteSource before being published to Azure Artifacts.

9. Conclusion

Summary of Key Points

In this guide, we have explored Azure Artifacts in depth, covering everything from setting up feeds to managing packages and integrating with CI/CD pipelines. We also discussed advanced features such as upstream sources, retention policies, and package versioning, and provided hands-on exercises to help you apply what you have learned.

By following the best practices outlined, you can ensure that your use of Azure Artifacts is efficient, secure, and scalable, enabling your teams to collaborate effectively and deliver high-quality software.

Further Reading and Resources

- Azure Artifacts Documentation
- Package Management with Azure Artifacts
- Azure DevOps Blog
- Microsoft Learn: Azure Artifacts