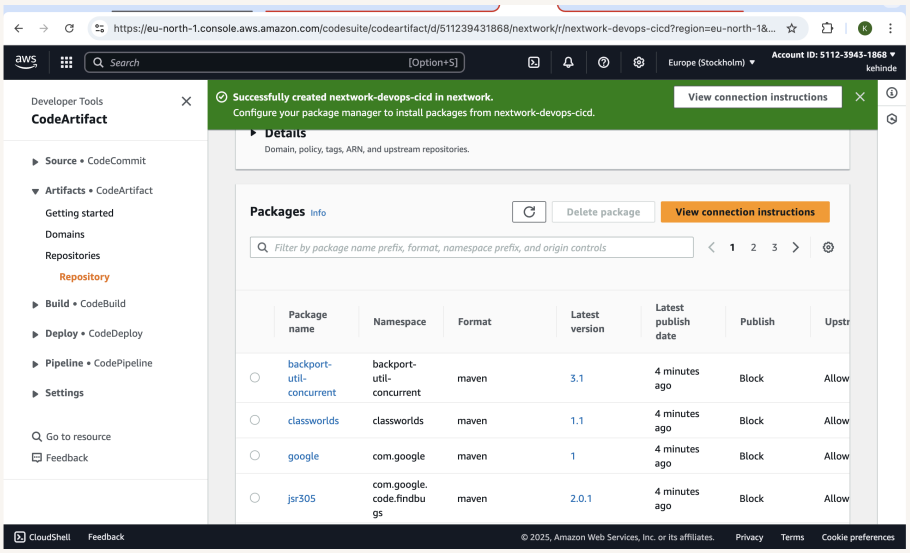
nextwork.org

Secure Packages with CodeArtifact

Kehinde Abiuwa



Introducing Today's Project!

In this project, I will demonstrate how to secure my web app's packages with AWS CodeArtifact. I'm doing this project to learn how to build a complete CI/CD pipeline that takes code from commit to production, while gaining practical DevOps skills that are highly valued in the tech and cloud industry.

Key tools and concepts

Services I used were Amazon CodeArtifact, AWS IAM, Amazon EC2, AWS CloudShell, and Maven. Key concepts I learnt include how to: Create and configure CodeArtifact domains and repositories to manage software packages securely. Use upstream repositories to pull dependencies from external sources when not available locally. Authenticate to CodeArtifact using authorization tokens and IAM roles instead of long-term credentials. Attach IAM policies and roles to EC2 instances for secure, role-based access. Configure Maven's settings.xml to work with private repositories in CodeArtifact. Publish and retrieve both public dependencies and custom, internal packages. Validate package integrity using CLI commands and checksum hashes. Apply DevOps principles to integrate package management into a CI/CD pipeline.

Project reflection

This project took me approximately 68 minutes. The most challenging part was troubleshooting the authorization token error — figuring out that I needed to create a new IAM policy, attach it to a role, and then assign that role to my EC2 instance before CodeArtifact access would work. It was most rewarding to see my own custom package successfully published to CodeArtifact and then retrieved, validated, and unpacked — proving end-to-end that my private package management workflow was working exactly as intended.

This project is part three of a series of DevOps projects where I'm building a CI/CD pipeline! I'll be working on the next project right away

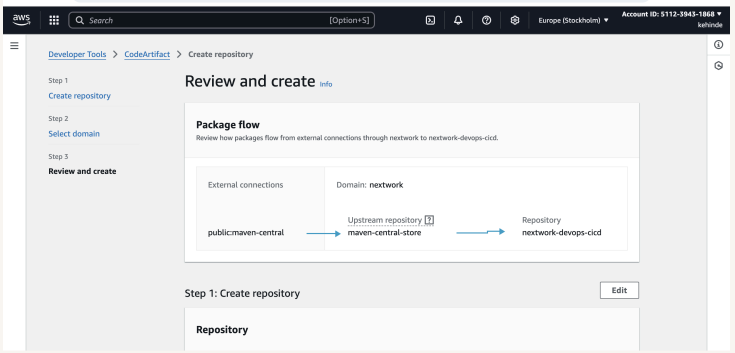
CodeArtifact Repository

CodeArtifact is an AWS-managed artifact repository service that provides a secure, centralized location for storing and sharing software packages and dependencies used in application development. Engineering teams use artifact repositories because they improve security by avoiding unverified public sources, increase reliability by providing a consistent and always-available source for dependencies, and enhance control by ensuring everyone uses the same approved versions of packages.

A domain is a top-level container in AWS CodeArtifact that can hold multiple repositories for a project or organization. It lets you apply permissions, access controls, and security settings once at the domain level, and those settings automatically apply to all repositories inside it. This makes managing multiple repositories more efficient and consistent. My domain is a centralized space in CodeArtifact where I manage all my repositories for this project, ensuring consistent security and access settings across them.

A CodeArtifact repository can have an upstream repository, which means it can pull packages from another repository (or a public package source) if the package isn't available locally. This ensures I still get the dependencies I need without manually searching for them elsewhere. My repository's upstream repository is the public Maven Central repository, which serves as a backup source for Java packages that aren't stored in my primary CodeArtifact repository.

Kehinde Abiuwa
NextWork Student



Kehinde Abiuwa
NextWork Student

CodeArtifact Security

Issue

To access CodeArtifact, we need an authorization token because CodeArtifact requires secure authentication before allowing us to view, download, or publish packages. The token proves to AWS that we have the correct permissions to access the repository. I ran into an error when retrieving a token because, by default, my EC2 instance doesn't have permission to access my other AWS services (including CodeArtifact). This is intentional - AWS follows the "principle of least privilege," meaning resources only get the minimum permissions they need to function.

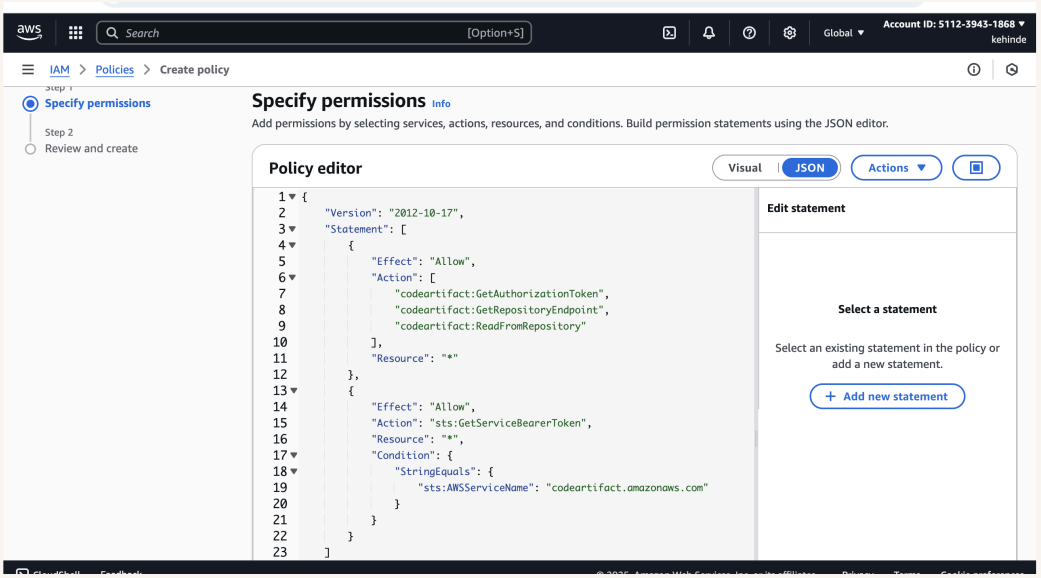
Resolution

To resolve the error with my security token, I created a new IAM policy with the necessary CodeArtifact permissions, attached that policy to a new IAM role, and then attached the role to my EC2 instance. This resolved the error because the EC2 instance could now assume the IAM role and automatically use its temporary credentials to authenticate with CodeArtifact, eliminating the need for manual token setup or locally stored credentials.

It’s security best practice to use IAM roles because they provide temporary, automatically rotated credentials instead of long-lived access keys, reducing the risk of credential leaks. IAM roles also let you grant only the minimum permissions needed for a specific task and can be assigned to AWS services or users dynamically, making access control more secure and easier to manage.

The JSON policy attached to my role

The JSON policy I set up grants permissions to: codeartifact:GetAuthorizationToken – lets me request an authentication token to securely connect to a CodeArtifact repository. codeartifact:GetRepositoryEndpoint – allows me to get the specific endpoint URL for a repository so my tools know where to send requests. codeartifact:ReadFromRepository – gives me permission to download packages from the repository. sts:GetServiceBearerToken (with a condition restricting it to codeartifact.amazonaws.com) – allows me to obtain a service bearer token for accessing CodeArtifact via AWS STS. These permissions are necessary because without them, I wouldn't be able to authenticate, locate, or retrieve packages from CodeArtifact securely. They give just enough access for reading from the repository without allowing destructive actions like deleting packages.



K
Kehinde Abiuwa
NextWork Student

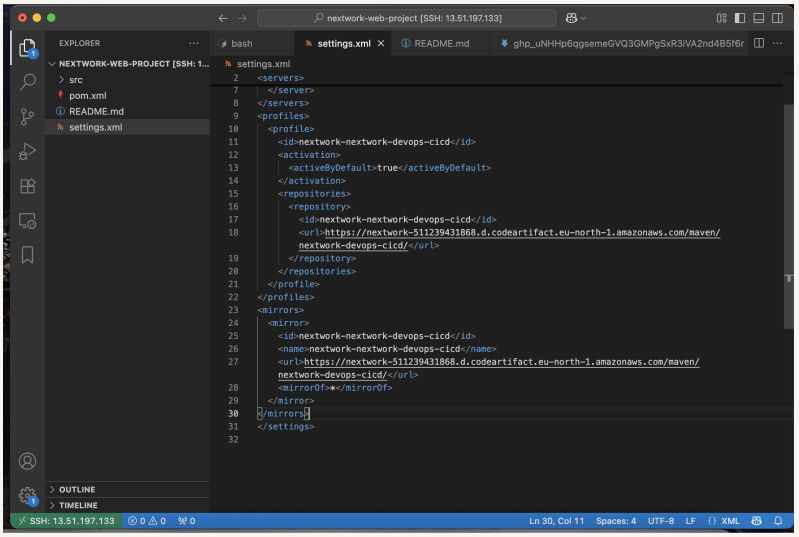
Maven and CodeArtifact

To test the connection between Maven and CodeArtifact, I compiled my web app using settings.xml

The settings.xml file configures Maven to authenticate with AWS CodeArtifact and use it as the primary repository for downloading and managing dependencies. In the <servers> section, it defines the repository ID, sets the username to aws, and uses the environment variable ${env.CODEARTIFACT_AUTH_TOKEN} for the password. This ensures secure, temporary authentication without storing plain-text credentials. In the <profiles> section, it sets up a profile with the same repository ID and URL of the CodeArtifact repository so Maven knows where to look for packages. In the <mirrors> section, it tells Maven to use the CodeArtifact repository as a mirror for all dependency requests (<mirrorOf>*</mirrorOf>), making it the default source for dependencies. This setup allows Maven to securely pull packages from CodeArtifact instead of relying on public repositories

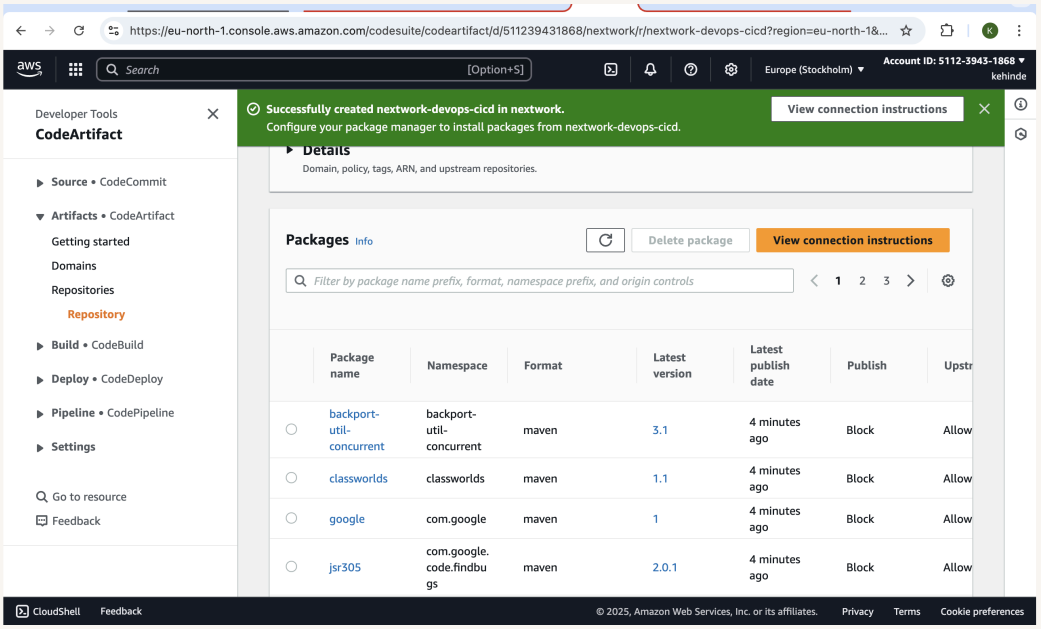
Compiling means translating human-readable source code written in a programming language (like Java) into machine-readable code (bytecode or binary) that a computer can execute. This process checks for syntax errors and ensures the code follows the rules of the language before producing an executable program or intermediate files.

Kehinde Abiuwa
NextWork Student



Verify Connection

After compiling, I checked my nextwork-devops-cicd repository in CodeArtifact and noticed that all the dependencies were downloaded into the repository from Maven. This happened because my Maven settings.xml was configured to use CodeArtifact as the central mirror, so any dependencies required for the build were fetched from Maven Central via CodeArtifact and then cached in my CodeArtifact repository for future use.



Kehinde Abiuwa
NextWork Student

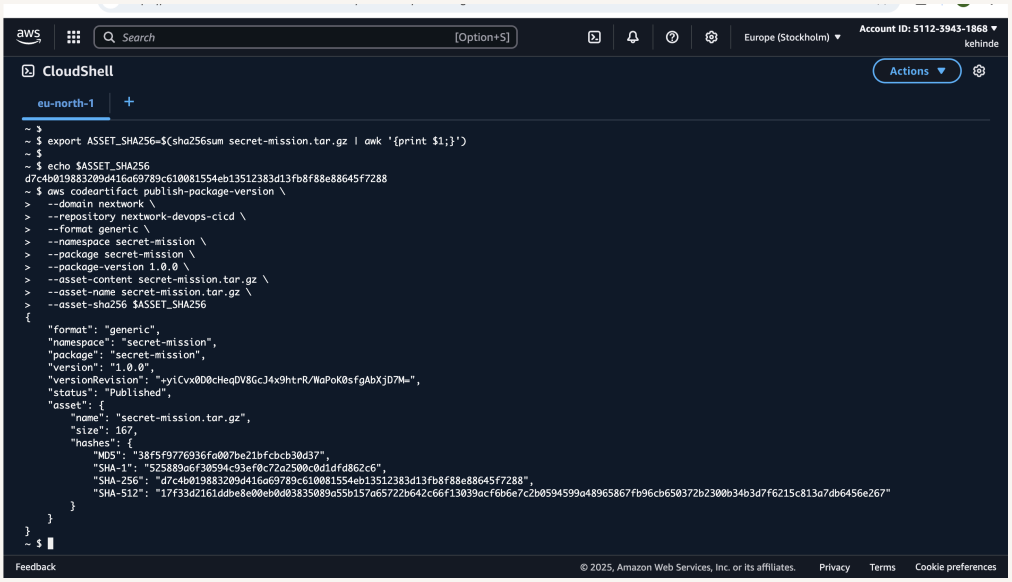
Uploading My Own Packages

In a project extension, I also decided to create my own custom package. This is useful in situations where I want to share proprietary, organization-specific code internally—such as shared utilities, common UI components, or specialized libraries—while keeping it private and secure within my company's AWS CodeArtifact repository, away from public access.

To create my own package, I first made a text file (secret-mission.txt) containing the message "Hellooooo this is a test package!". I then compressed it into a .tar.gz archive (secret-mission.tar.gz) using the tar -czvf command so it could be uploaded and distributed as a single package file. I also generated a security hash because the SHA-256 checksum uniquely identifies the exact contents of the package. This allows CodeArtifact (or any other system) to verify the package's integrity during upload or download, ensuring that the file hasn't been altered or corrupted.

To publish the package, I uploaded my secret-mission archive to my nextwork-devops-cicd CodeArtifact repository, specifying the package name, version (1.0.0), and type. When I look at the package details in CodeArtifact, I can see that the package secret-mission is stored in my repository, its latest version is 1.0.0, it has the status "Published," the origin is nextwork-devops-cicd, and it's marked as an internal package—meaning it's private to my organization.

To validate my packages, I then used the aws codeartifact get-package-version-asset command to download the secret-mission package (1.0.0) from my nextwork-devops-cicd repository, extracted it with tar -zxvf, and opened the file inside. I saw that the extracted file secret-mission.txt contained the message: Hellooooo this is a test package! This confirmed that my package was successfully published to and retrieved from CodeArtifact without corruption.



The place to learn & showcase your skills

Check out nextwork.org for more projects