PHASE 2

CODEARTIFACT SETUP

STEP 1 : SECURE PACKAGES WITH CODEARTIFACT

Definitions:

AWS CodeArtifact: It is an artifact repository , where you store all of the app's packages in one place. An important part of CI/CD pipeline as it makes the engineering team make sure that they are using the same ,verified versions of packages when building and deploying the app.

- Head to AWS Management Console→ Code Artifact→ Create repository
- Give a name for repo and description , for Public upstream repositories select PyPI

  Note: Why Set PyPI as Upstream?

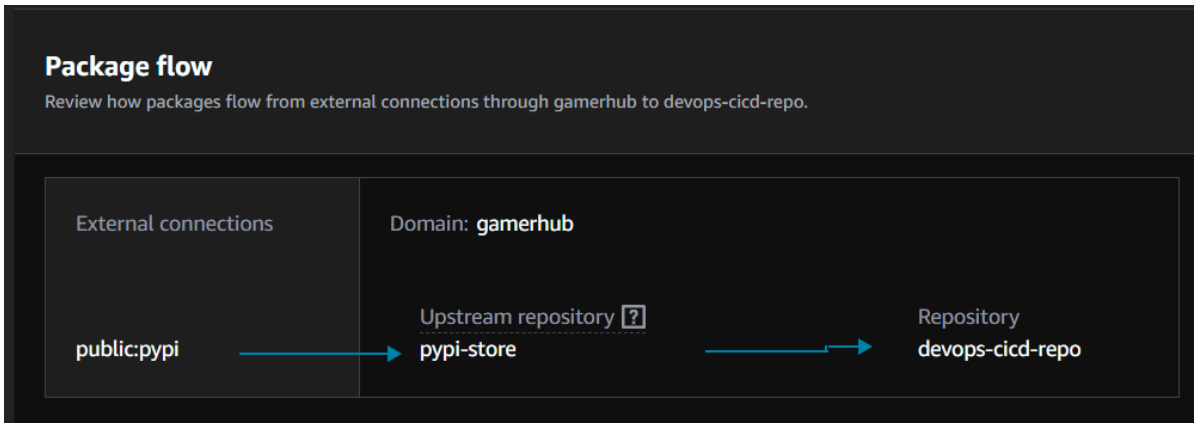
  Reason for doing that:

  - **Speed:** The first time Flask or SQLAlchemy is installed, it's fetched from PyPI → then cached in CodeArtifact. Next time, it's lightning fast
  - **Resilience:** Even if PyPI is down, CodeArtifact gotchu.
  - **Audit/Control:** You can monitor and restrict external packages like `tensorflow==666.0.0` if it breaks everything
  - It is more likely that the Central Repository (PyPI) goes down sometimes. So to avoid any issue we cache the libraries in CodeArtifact.

- Click Next → Domain to be selected

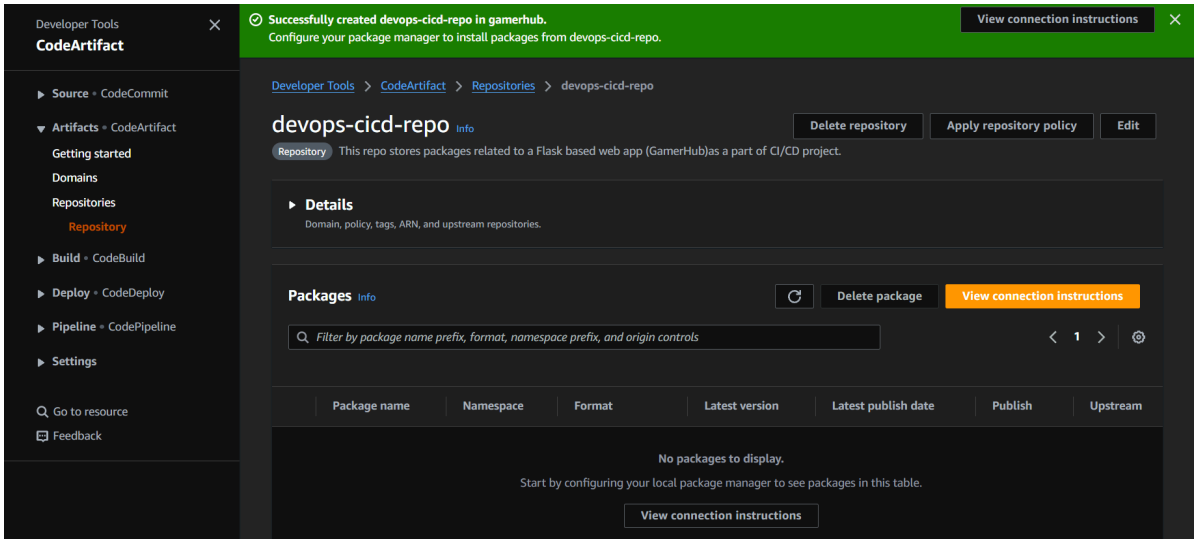
> **What is a CodeArtifact domain?**
>
> A CodeArtifact domain is like a folder that holds multiple repositories belonging to the same project or organization. We like using domains because they give you a single place to manage permissions and security settings that apply to *all* repositories inside it. This is much more convenient than setting up permissions for each repository separately, especially in large companies where many teams need access to different repositories.
>
> With domains, you can ensure consistent security controls across all your package repositories in an efficient way.
>
> **Analogy:**
>
> | AWS Thing | Real-world Equivalent |
> | --- | --- |
> | CodeArtifact Domain | Your warehouse building 🏢 |
> | Repository | Specific shelf inside it 🧰 |
> | Upstream (PyPI) | External supplier 🛒 |

- In Domain tab , select This AWS Account and give a domain name → Click Next



> What is this package flow diagram?
>
> The package flow diagram shows you exactly how dependencies will travel to your application. When your project needs a dependency, it first looks in your CodeArtifact repository. If the package is already there, great! It uses that version. If not, CodeArtifact automatically reaches out to Maven Central to fetch it.
>
> This is important to understand because it affects how quickly your builds run and how resilient they are to network issues. The first time you request a package, it might take a moment longer as CodeArtifact fetches it from Maven Central. But every build afterwards will be faster because the package is now cached in your repository. It's like the difference between ordering groceries for delivery versus already having them in your fridge!

- Select create repository



The repo is created successfully

STEP 2: Create an IAM Policy for CodeArtifact Access

For PyPI to work with CodeArtifact , an IAM role is to be created to grant the EC2 instance the permission to access CodeArtifact.

If an IAM role is not created :

- Click View connection instructions → Selecting proper OS . Choose package management client as pip , select configuration method as Manual method: pull from your repository with pip.
- Copy command in step 3 → Go to EC2 instance session (VSCode or MobaXterm) and paste the command.
- We get an error....

```
(.venv) [ec2-user@ip-172-31-87-169 GamerHub]$ export CODEARTIFACT_AUTH_TOKEN=`aws codeartifact get-authorization-token --domain gamerhub --domain-owner 975050
195505 --region us-east-1 --query authorizationToken --output text`

Unable to locate credentials. You can configure credentials by running "aws configure".
(.venv) [ec2-user@ip-172-31-87-169 GamerHub]$
```

- This is actually a good security feature . Here the EC2 instance doesn't identify the user with proper authorization so it doesn't let the user access CodeArtifact. Happens because by default EC2 instances doesn't have permission to access other AWS services by the principle of least privilege.

So to create an IAM role :

- Go to IAM → Policies → Create policies → JSON tab and paste the following JSON code

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "codeartifact:GetAuthorizationToken",
            "codeartifact:GetRepositoryEndpoint",
            "codeartifact:ReadFromRepository"
         ],
         "Resource": "*"
      },
      {
         "Effect": "Allow",
         "Action": "sts:GetServiceBearerToken",
         "Resource": "*",
         "Condition": {
            "StringEquals": {
               "sts:AWSServiceName": "codeartifact.amazonaws.com"
            }
         }
      }
   ]
}
```

In this JSON document:

1. The first part (codeartifact:* actions) gives permission to get authentication tokens, find repository locations, and read packages from repositories.
2. The second part (sts:GetServiceBearerToken) allows temporarily elevated access specifically for CodeArtifact operations.

The "Resource": "*" means these permissions apply to all relevant resources, while the Condition narrows the second permission to only work with CodeArtifact.

This follows the "principle of least privilege" - granting only the minimum permissions needed to perform the required tasks, enhancing your security posture.

DeepDive :

> What does this JSON policy document do?
>
> This JSON document defines an IAM policy that grants specific permissions required for accessing AWS
>
> CodeArtifact repositories. Let's break down the policy:

- **Version**: "2012-10-17" is the version of the policy language.
- **Statement** is a list of statement objects, each defining one or more permissions.

First Statement:

- "Effect": "Allow" means that this statement grants permission.
- "Action" lists the actions that are allowed. Here, it includes:
    - "codeartifact:GetAuthorizationToken" allows getting an authorization token for CodeArtifact.
    - "codeartifact:GetRepositoryEndpoint" allows retrieving the endpoint for a CodeArtifact repository.
    - "codeartifact:ReadFromRepository" allows reading packages from a CodeArtifact repository.

- "Resource": "*" means these actions are allowed on all resources (*) in CodeArtifact. In a production environment, you would typically restrict this to specific resources for better security.

Second Statement:

- "Effect": "Allow" means this statement grants permission.
- "Action": "sts:GetServiceBearerToken" allows calling the GetServiceBearerToken action from the AWS Security Token Service (STS).
- "Resource": "*" means this action is allowed on all resources.
- "Condition" adds a condition to this permission.
    - "StringEquals" is a condition that checks for string equality.
    - "sts:AWSServiceName": "codeartifact.amazonaws.com": This

> condition ensures that the sts:GetServiceBearerToken action is only allowed when the AWS service name is codeartifact.amazonaws.com. This is a security measure to restrict the use of this STS action specifically for CodeArtifact.

- Give a name to the policy and click on Create policy
- Go to Roles under IAM. Create a new Role by attaching the above created policy.
- Attach the IAM role → Go to Instances → Select instance → Actions → Security → Modify IAM Role → Attach the role and confirm
- Rerun the export token command from step 3 earlier in EC2 session.
- export CODEARTIFACT_AUTH_TOKEN=”.....” → This command will retrieve a temporary authorization token for CodeArtifact and store it in an environment variable named CODEARTIFACT_AUTH_TOKEN.
- If above command still shows error, then reboot the instance and try again.

STEP 3: See Packages in CodeArtifact

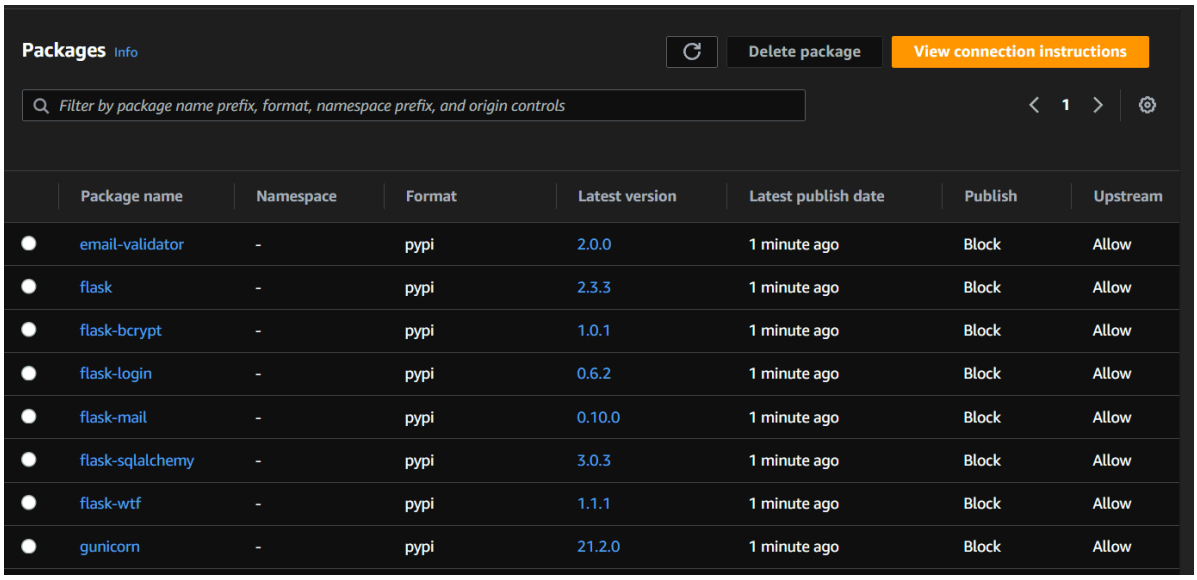
- Go to the codeartifact repo , choose Linux or MAC , Step 3 is already done. Do step 4.

  pip config set global.index-url https://aws:$CODEARTIFACT_AUTH_TOKEN@gamerhub-

975050195505.d.codeartifact.us-east-1.amazonaws.com/pypi/devops-cicd-repo/simple/

- That command wrote this to `/home/ec2-user/.config/pip/pip.conf.` Now whenever you run **any** `pip install`, it uses **your CodeArtifact repo** by default.
- Then do:

  pip install --extra-index-url https://aws:$CODEARTIFACT_AUTH_TOKEN@gamerhub-975050195505.d.codeartifact.us-east-1.amazonaws.com/pypi/devops-cicd-repo/simple/ -r requirements.txt
- Checking status: Head to the CodeArtifact window → Refresh the packages



Why Are Packages Showing Up in CodeArtifact?

The appearance of packages in your AWS CodeArtifact repository confirms that the integration between your Python environment and CodeArtifact is functioning correctly.

Here's what happens behind the scenes when you run a `pip install` command:

1. **Dependency Detection**

   Pip reads your `requirements.txt` file to determine which packages your Python application needs.

2. **Package Request Flow**

   Since you configured CodeArtifact as your package index (via `pip.conf` or the `--extra-index-url` flag), pip requests these dependencies from your CodeArtifact repository instead of directly from PyPI.

3. **First-Time Fetching**

   If the requested packages are not yet stored in your CodeArtifact repository, CodeArtifact automatically checks its upstream repository (PyPI) to retrieve them.

4. **Storage and Caching**

   After fetching the dependencies from PyPI, CodeArtifact stores local copies in your repository. These packages are then returned to your pip installation process.

5. **Subsequent Requests**

   On future builds or when other developers in your organization install the same packages, CodeArtifact serves them directly from its cache, improving speed and reliability.

6. **Centralized Control**

   This setup allows your organization to maintain consistent dependency versions, enforce package policies, and operate independently of PyPI availability.

_____PHASE 2 COMPLETED_____