Picture 1. Global Pipeline Workflow

Create a new service account in Google Cloud IAM

The first thing we must do is create Service Account in Google Cloud IAM, Service Account functions as a restriction on access to resources we use later. We can use this command:

```
gcloud projects list # to get project ID
export PROJECT_ID="your-project-id" # Set your project id in here   set env variable
```

after that, we run this command to create the Service Account:

```
gcloud iam service-accounts create "cloud-run-sa" --project="${PROJECT_ID}"
--description="Cloud Run Service Account" --display-name="Cloud Run Service
Account"
```

We set roles as Artifact Registry Admin and Cloud Run Admin because we need access to those resources, we can run this command:

```
gcloud projects add-iam-policy-binding "${PROJECT_ID}"
--member="serviceAccount:cloud-run-sa@${PROJECT_ID}.iam.gserviceaccount.com
" --role="roles/artifactregistry.repoAdmin" --role="roles/run.admin"
--role="roles/iam.serviceAccountUser"
```

```
rofim58@cloudshell:~ (bangkit-project-150424)$ export PROJECT_ID="bangkit-project-150424"
rofim58@cloudshell:~ (bangkit-project-150424)$ gcloud iam service-accounts create \
    "cloud-run-sa" \
    --project="${PROJECT_ID}" \
    --description="Cloud Run Service Account" \
    --display-name="Cloud Run Service Account"
Created service account [cloud-run-sa].
```

Picture 2. Run the command to create the service account

```
rofim58@cloudshell:~ (bangkit-project-150424)$ gcloud projects add-iam-policy-binding "${PROJECT_ID}" \
    --member="serviceAccount:cloud-run-sa@${PROJECT_ID}.iam.gserviceaccount.com" \
    --role="roles/artifactregistry.repoAdmin" \
    --role="roles/run.admin"
Updated IAM policy for project [bangkit-project-150424].
bindings:
- members:
  - serviceAccount:service-223643940089@gcp-gae-service.iam.gserviceaccount.com
  role: roles/appengine.serviceAgent
- members:
  - serviceAccount:service-223643940089@gcp-sa-artifactregistry.iam.gserviceaccount.com
  role: roles/artifactregistry.serviceAgent
- members:
  - serviceAccount:223643940089@cloudbuild.gserviceaccount.com
  role: roles/cloudbuild.builds.builder
- members:
  - serviceAccount:service-223643940089@gcp-sa-cloudbuild.iam.gserviceaccount.com
  role: roles/cloudbuild.serviceAgent
- members:
  - serviceAccount:service-223643940089@compute-system.iam.gserviceaccount.com
  role: roles/compute.serviceAgent
- members:
  - serviceAccount:service-223643940089@container-engine-robot.iam.gserviceaccount.com
  role: roles/container.serviceAgent
- members:
  - serviceAccount:service-223643940089@containerregistry.iam.gserviceaccount.com
  role: roles/containerregistry.ServiceAgent
- members:
  - serviceAccount:223643940089-compute@developer.gserviceaccount.com
  - serviceAccount:223643940089@cloudservices.gserviceaccount.com
  - serviceAccount:bangkit-project-150424@appspot.gserviceaccount.com
  role: roles/editor
- members:
  - serviceAccount:service-223643940089@gcp-sa-firestore.iam.gserviceaccount.com
  role: roles/firestore.serviceAgent
- members:
  - serviceAccount:service-223643940089@gcp-sa-networkconnectivity.iam.gserviceaccount.com
  role: roles/networkconnectivity.serviceAgent
- members:
  - user:rofim58@gmail.com
  role: roles/owner
- members:
  - serviceAccount:service-223643940089@gcp-sa-pubsub.iam.gserviceaccount.com
  role: roles/pubsub.serviceAgent
- members:
  - serviceAccount:cloud-run-sa@bangkit-project-150424.iam.gserviceaccount.com
```

Picture 3. Run the command to assign a role to the service account

	Email	Status	Name ↑	Description	Key ID	Key creation date	OAuth 2 Client ID	Actions
☐								
☐	cloud-run-sa@bangkit-project-150424.iam.gserviceaccount.com	✓ Enabled	Cloud Run Service Account	Cloud Run Service Account	No keys			⋮
☐								

https://console.cloud.google.com/iam-admin/serviceaccounts

Picture 4. Result service account

Create a new Workload Identity Pool

A workload identity pool is an entity that lets you manage external identities.

First, we must export the repo owner and repo name

```
export REPO_OWNER="your-github-username"
export REPO_NAME="your-repo-name"
```

We create a Workload Identity Pool we named "github" using this command:

```
gcloud iam workload-identity-pools create "github" \
  --project="${PROJECT_ID}" \
  --location="global" \
  --display-name="GitHub Actions Pool"
```

We can check WIP was successfully created or not using this command:

```
gcloud iam workload-identity-pools describe "github" \
  --project="${PROJECT_ID}" \
  --location="global" \
  --format="value(name)"
```

We create OpenID Connect for an identity provider, This command sets up an OIDC identity provider that allows Google Cloud to trust tokens issued by GitHub Actions. By mapping specific attributes from the GitHub OIDC token, it enables fine-grained access control based on these attributes in your Google Cloud environment:

```
gcloud iam workload-identity-pools providers create-oidc
"github-repo-provider" \
  --project="${PROJECT_ID}" \
  --location="global" \
  --workload-identity-pool="github" \
  --display-name="My GitHub repo Provider" \
--attribute-mapping="google.subject=assertion.sub,attribute.actor=assertion
.actor,attribute.repository=assertion.repository,attribute.repository_owner
=assertion.repository_owner,attribute.repository_id=assertion.repository_id
" \
  --issuer-uri="https://token.actions.githubusercontent.com"
```

This sequence of commands allows a specific GitHub repository to assume the cloud-run-sa service account's identity, enabling it to interact with Google Cloud

resources as this service account. This setup is useful for securely granting permissions to GitHub Actions workflows to interact with Google Cloud.
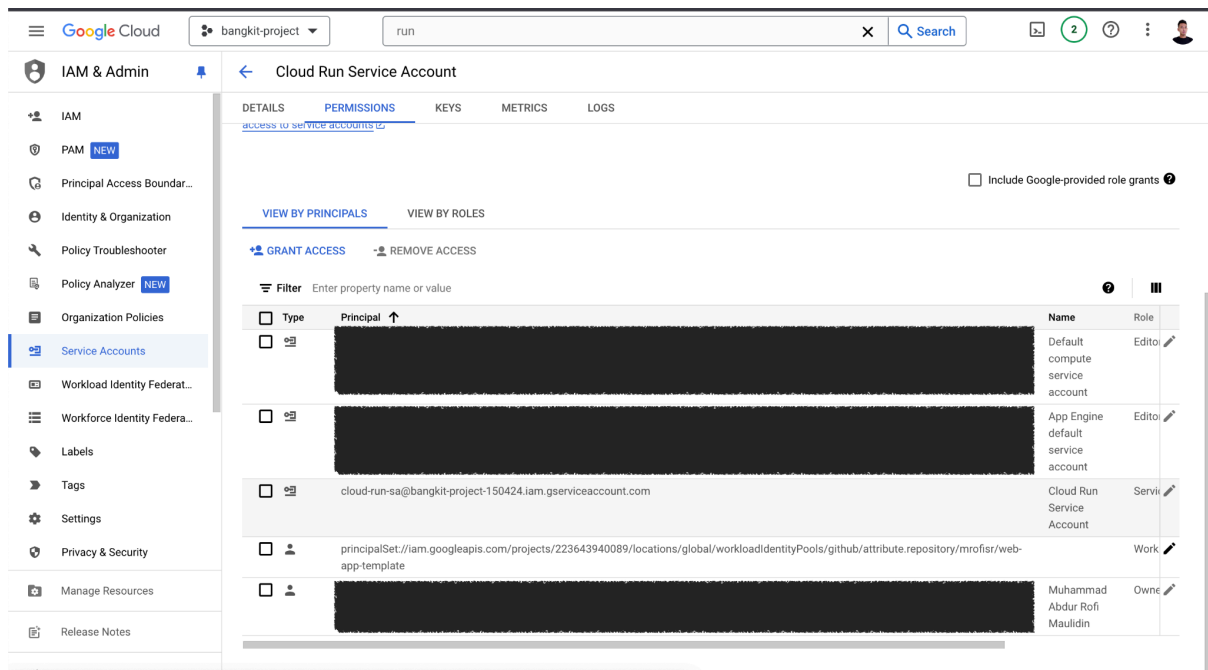
```
export SA_EMAIL="cloud-run-sa@${PROJECT_ID}.iam.gserviceaccount.com"
export WORKLOAD_POOL=`gcloud iam workload-identity-pools describe "github" \
  --project="${PROJECT_ID}" \
  --location="global" \
  --format="value(name)"`

gcloud iam service-accounts add-iam-policy-binding ${SA_EMAIL}
--project="${PROJECT_ID}" --role="roles/iam.workloadIdentityUser"
--member="principalSet://iam.googleapis.com/${WORKLOAD_POOL}/attribute.repository/${REPO_OWNER}/${REPO_NAME}"
```


*Picture 4. Result in Workload Identity Pool and Provider*

*Picture 5. Result in service account assign member and roles*

Setup GitHub Actions Secret

To setup GitHub Actions Secret we need 3 items:

- PROJECT_ID, use this command to get the project ID `gcloud projects list`
- SERVICE_ACCOUNT, value is the email of the service account
  `cloud-run-sa@${PROJECT_ID}.iam.gserviceaccount.com`
- WORKLOAD_IDENTITY_PROVIDER, use this command to get WIP `gcloud iam workload-identity-pools providers describe "github-repo-provider" --project="${PROJECT_ID}" --location="global" --workload-identity-pool="github" --format="value(name)"`
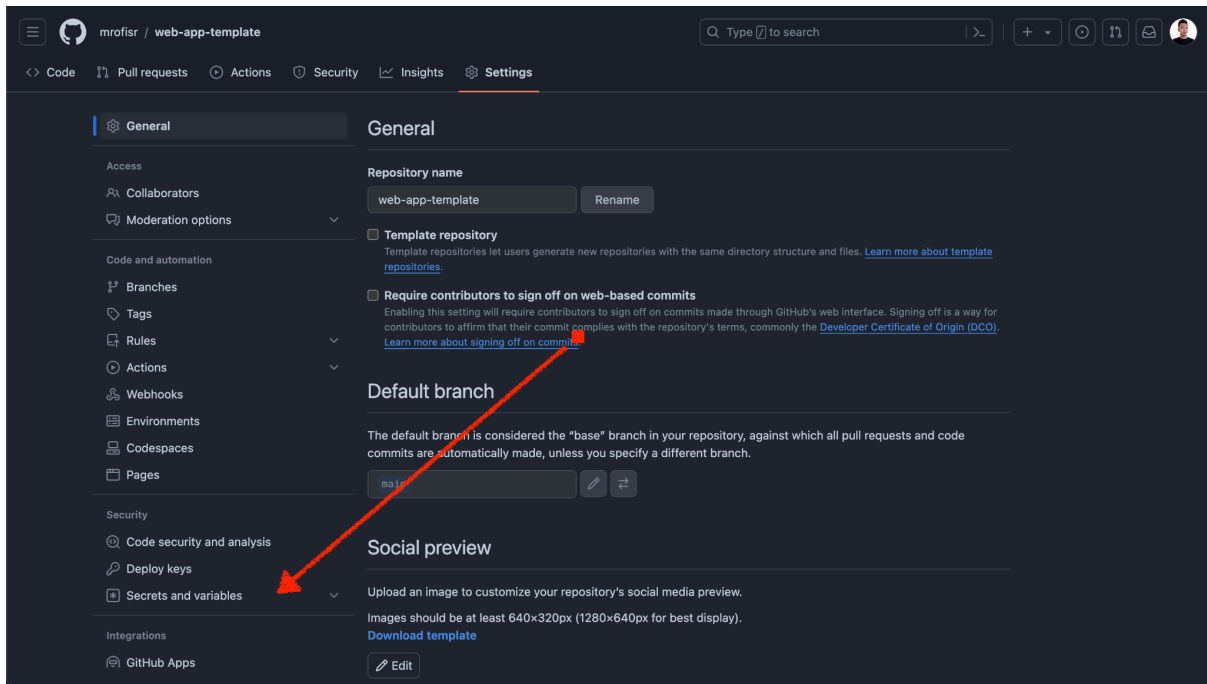
Picture 6. Create GitHub Actions secret
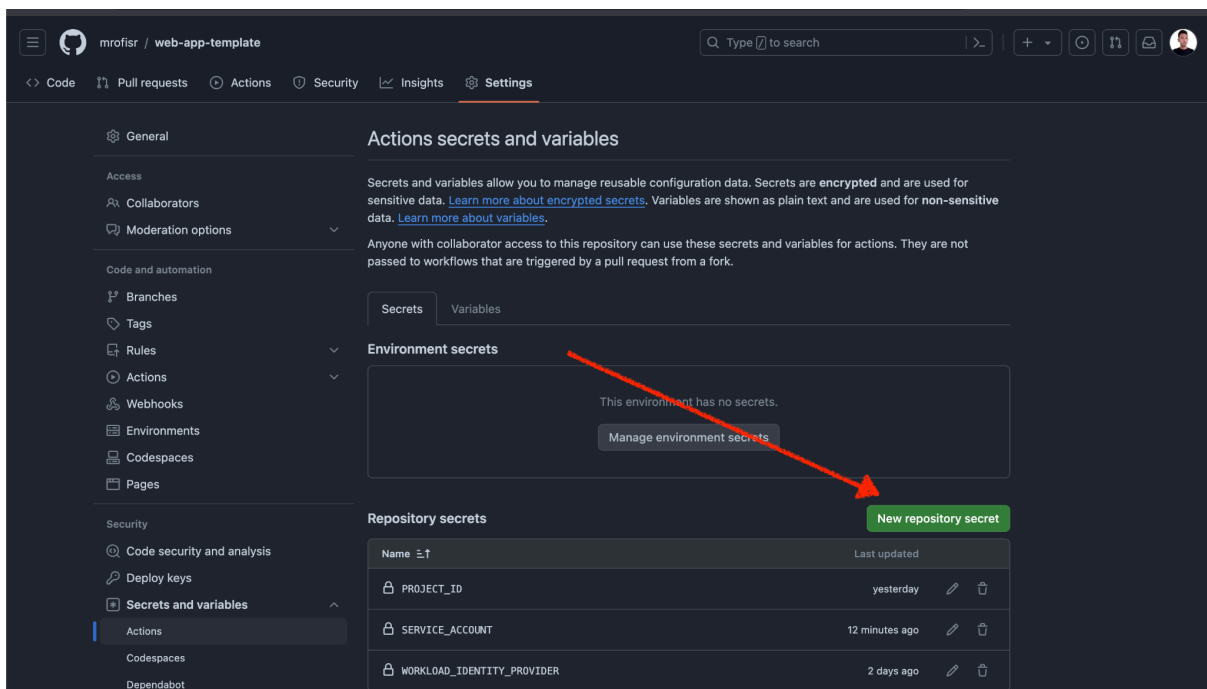
Picture 7. Add new GitHub Actions secret
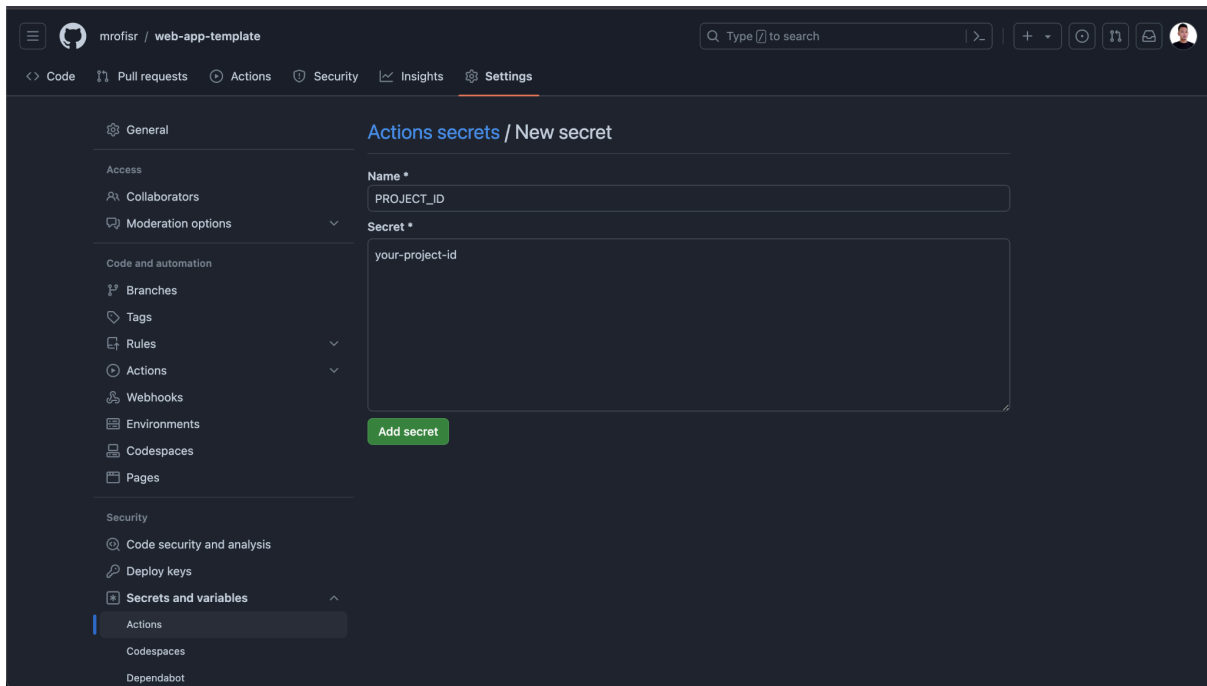
Picture 8. Set key value GitHub Actions secret

Setup Pipeline

First, we must create a directory `.github/workflows` and create new file inside the directory named `deployment.yaml` and fill it with this:

```yaml
name: CI and CD Pipeline to Cloud Run

on:
  push:
    branches:
      - main
      - dev
  pull_request:
    branches:
      - main
      - dev
    types:
      - closed

jobs:
  build-and-deploy:
    permissions:
      contents: 'read'
      id-token: 'write'
    runs-on: ubuntu-latest
    steps:
      # Step to checkout the code from the repository
      - name: Checkout code
        id: checkout
```

```yaml
      uses: actions/checkout@v4

    # Step to set environment variables
    - name: Set environment variables
      id: set-env
      run: |
        echo "IMAGE_VERSION=${{ github.ref_name }}" >> $GITHUB_ENV

    # Step to authenticate with Google Cloud
    - name: Gcloud Auth
      id: gcloud-auth
      uses: google-github-actions/auth@v2
      with:
        token_format: 'access_token'
        project_id: ${{ secrets.PROJECT_ID }}
        service_account: ${{ secrets.SERVICE_ACCOUNT }}
        workload_identity_provider: ${{
secrets.WORKLOAD_IDENTITY_PROVIDER }}

    # Step to authenticate with the Google Container Registry
    - name: Auth Container Registry
      id: container-registry-auth
      uses: docker/login-action@v3
      with:
        registry: us-central1-docker.pkg.dev
        username: oauth2accesstoken
        password: ${{ steps.gcloud-auth.outputs.access_token }}

    # Step to build and push the Docker image for the backend
    - name: Build and push Docker image Backend
      id: build-push-back
      uses: docker/build-push-action@v6
      with:
        context: .
        file: ./Dockerfile.back
        push: true
        tags: us-central1-docker.pkg.dev/${{ secrets.PROJECT_ID
}}/cloud-run/cloud-run-back:${{ env.IMAGE_VERSION }}

    # Step to deploy the backend Docker image to Cloud Run
    - name: Deploy to Cloud Run Backend
      id: deploy-cloud-run-back
      uses: google-github-actions/deploy-cloudrun@v2
      with:
        service: cloud-run-backend-${{ env.IMAGE_VERSION }}
        image: us-central1-docker.pkg.dev/${{ secrets.PROJECT_ID
}}/cloud-run/cloud-run-back:${{ env.IMAGE_VERSION }}
        region: us-central1
        project_id: ${{ secrets.PROJECT_ID }}
        flags: --port 4000 --allow-unauthenticated

    # Step to update the NGINX configuration with the new backend URL
```

```
    - name: Change backend stream NGINX config
      run: |
        DOMAIN=$(echo ${{ steps.deploy-cloud-run-back.outputs.url }} |
cut -d'/' -f3)
        sed -i 's|backend:4000|'${DOMAIN}'|g' ./nginx.conf
        sed -i 's|http://backend|https://'${DOMAIN}'|g' ./nginx.conf
        cat ./nginx.conf

    # Step to build and push the Docker image for the frontend
    - name: Build and push Docker image Frontend
      id: build-push-front
      uses: docker/build-push-action@v6
      with:
        context: .
        file: ./Dockerfile.front
        push: true
        tags: us-central1-docker.pkg.dev/${{ secrets.PROJECT_ID
}}/cloud-run/cloud-run-front:${{ env.IMAGE_VERSION }}

    # Step to deploy the frontend Docker image to Cloud Run
    - name: Deploy to Cloud Run Frontend
      id: deploy-cloud-run-front
      uses: google-github-actions/deploy-cloudrun@v2
      with:
        service: cloud-run-frontend-${{ env.IMAGE_VERSION }}
        image: us-central1-docker.pkg.dev/${{ secrets.PROJECT_ID
}}/cloud-run/cloud-run-front:${{ env.IMAGE_VERSION }}
        region: us-central1
        project_id: ${{ secrets.PROJECT_ID }}
        flags: --port 80 --allow-unauthenticated

    # Step to test if both the backend and frontend are running correctly
    - name: Test Cloud Run
      run: |
        curl "${{ steps.deploy-cloud-run-back.outputs.url }}" -I -s |
grep "HTTP/2 200"
        curl "${{ steps.deploy-cloud-run-front.outputs.url }}" -I -s |
grep "HTTP/2 200"
```

Test Pipeline

youtu.be/1BGgfenYUhE