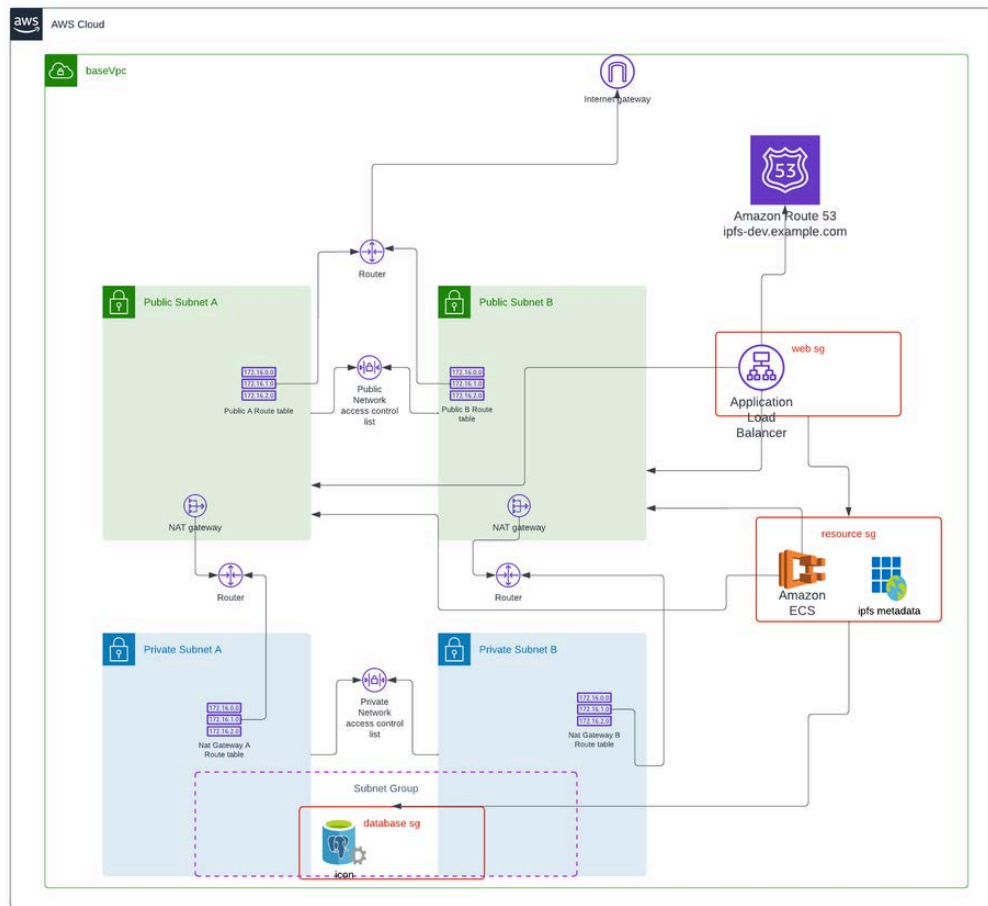


IPFS Metadata Architecture and Deployment



ipfs aws architecture

Architecture Overview

This infrastructure is deployed on AWS, leveraging a combination of public and private subnets across multiple availability zones to ensure high availability, security, and cost efficiency.

The architecture consists of two public subnets and two private subnets, strategically placed in different availability zones to meet AWS's requirement for subnet redundancy. This design is aimed at optimizing cost without compromising on security or functionality. The database is securely housed within the private subnets, protecting it from direct exposure to the internet, while essential resources such as containers and load balancers are deployed in the public subnets.

To control and secure the flow of traffic, three distinct security groups are utilized:

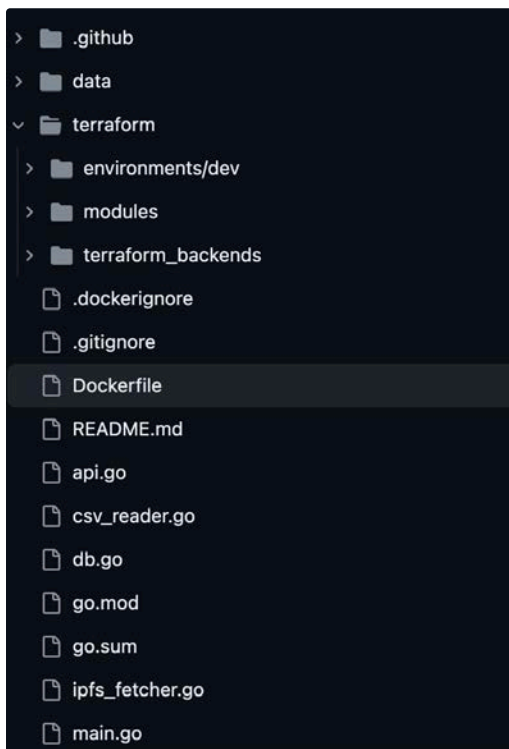
- **Web Security Group (web sg):** Manages inbound and outbound traffic for the web-facing components, particularly the Application Load Balancer.
- **Resource Security Group (resource sg):** Governs access to the ECS containers running the IPFS application, with proper health checks and target groups configured to maintain application availability and reliability.
- **Database Security Group (database sg):** Ensures that the RDS instance is securely accessed only by the authorized resources within the VPC.

The ECS containers hosting the IPFS application are deployed with the resource security group, ensuring that access is tightly controlled and monitored. The Application Load Balancer, positioned within the public subnets, distributes incoming traffic to the appropriate containers, ensuring a smooth and scalable operation.

Additionally, the architecture includes a NAT Gateway configured in each availability zone, providing secure and efficient outbound internet access for resources within the private subnets. This setup minimizes the load on the NAT Gateway, focusing its usage on essential traffic, such as the one directed towards the RDS instance.

It's important to note that AWS mandates the use of at least two subnets in different availability zones for services requiring high availability, such as RDS instances. Therefore, this architecture uses a minimum of two private subnets for the database and two public subnets for resources that require internet accessibility, ensuring compliance with AWS best practices.

IPFS Application Folder Structure Overview



- **.github/**: Contains GitHub-specific configuration files, such as workflows for continuous integration and deployment.
- **data/**: houses datasets or input files that the application needs to process or interact with.
- **terraform/**: This directory contains Terraform configuration files for managing the infrastructure as code.
 - **environments/dev/**: Holds environment-specific Terraform files, particularly for the development environment.
 - **modules/**: Contains reusable Terraform modules that define common configurations, promoting consistency and reusability across different environments.
 - **terraform_backends/**: Stores backend configurations, such as remote state storage settings for Terraform, ensuring state consistency across multiple users or machines.
- **.dockerignore**: Specifies files and directories that should be excluded from Docker builds, reducing image size and build time.
- **.gitignore**: Lists files and directories that Git should ignore, preventing unnecessary files from being tracked in version control.
- **Dockerfile**: Defines the instructions for building the Docker image for the application, specifying dependencies, build steps, and the entry point.
- **README.md**: Provides an overview of the project, including setup instructions, usage guidelines, and other relevant documentation.
- **api.go**: contains the Go code that defines the application's API endpoints.

- **csv_reader.go**: A Go file dedicated to reading and processing CSV files, possibly for input data handling.
- **db.go**: Manages database interactions, such as connecting to the database, executing queries, and handling data persistence.
- **go.mod**: Defines the dependencies and modules used by the Go application, ensuring consistent builds and dependency management.
- **go.sum**: Locks the versions of the dependencies listed in `go.mod`, providing repeatable builds by tracking checksums.
- **ipfs_fetcher.go**: handles interactions with IPFS (InterPlanetary File System), fetching or managing content from the distributed file system.
- **main.go**: The entry point of the Go application, where the main logic of the program is executed.

Deployment Workflows

Workflows Overview



This project employs a series of automated workflows, each defined in its own YAML file, to manage the infrastructure and application deployment processes on AWS. The workflows are organized to cover the full lifecycle of the IPFS Metadata infrastructure, from setup to teardown.

1. Setup Terraform State Backend (`setup-state-backend.yml`)

- **Action Name:** *01 Setup Terraform State Backend*
- **Purpose:** This workflow is responsible for setting up the Terraform state backend, which is crucial for managing the Terraform state files. It ensures that the state is securely stored and accessible across different deployments, preventing conflicts and ensuring consistency.

2. Plan IPFS Metadata Infrastructure (`pr.yml`)

- **Action Name:** *02 Plan IPFS Metadata Infrastructure*
- **Purpose:** This workflow handles the planning stage of the infrastructure deployment. It runs `terraform plan` to generate an execution plan, which outlines the changes that will be applied to the infrastructure. This step helps verify the changes before they are applied, ensuring that they align with expectations.

3. Deploy IPFS Metadata Infrastructure (`deploy-infrastructure.yml`)

- **Action Name:** *03 Deploy IPFS Metadata Infrastructure*
- **Purpose:** This workflow is responsible for deploying the IPFS Metadata infrastructure using Terraform. It applies the infrastructure changes defined in the Terraform scripts, provisioning the necessary AWS resources, such as VPCs, subnets, ECS clusters, and security groups.

4. Deploy IPFS Metadata Application (`deploy-ipfs-app.yml`)

- **Action Name:** *04 Deploy IPFS Metadata Application*
- **Purpose:** This workflow deploys the IPFS Metadata application to the infrastructure set up in the previous step. It involves building and deploying the Docker containers, and ensuring the application is running as expected within the configured environment.

5. Destroy IPFS Metadata Infrastructure (`destroy-infrastructure.yml`)

- **Action Name:** *05 Destroy IPFS Metadata Infrastructure*

- **Purpose:** This workflow is designed to tear down the entire IPFS Metadata infrastructure. It runs `terraform destroy`, which removes all the AWS resources provisioned for the project, ensuring a clean and cost-effective removal of the environment when it's no longer needed.

Configuration Overview for Deployment Workflows

Before initiating the deployment workflows, it is essential to configure the GitHub environment properly. This configuration sets the foundation for automating infrastructure and application deployments using GitHub Actions. Below is a detailed description of the necessary configuration:

1. GitHub Environments

- **Environments to Create:**

- **dev:** This is the primary environment where development and testing occur. It's mandatory to have this environment configured.
- **prod (optional):** If the application will run in production, this environment should be configured as well. However, it's optional if the production deployment is not required.

- **Environment Variables and Secrets:**

- **Secrets:**

- **SHARED_CONFIG:** This secret is a configuration string for the AWS CLI, specifying the profile and region.

- Example for `dev`:

```
1 [profile dev]\\nregion = us-east-2\\n
```

- Example for `prod`:

```
1 [profile prod]\\nregion = us-west-2\\n
```

- **SHARED_CREDENTIALS:** This secret contains the AWS credentials required for the environment. Replace `***` with the exact value for your environment.

- Example for `dev`:

```
1 [dev]\\naws_access_key_id = ***\\naws_secret_access_key = ***\\n
```

- Example for `prod`:

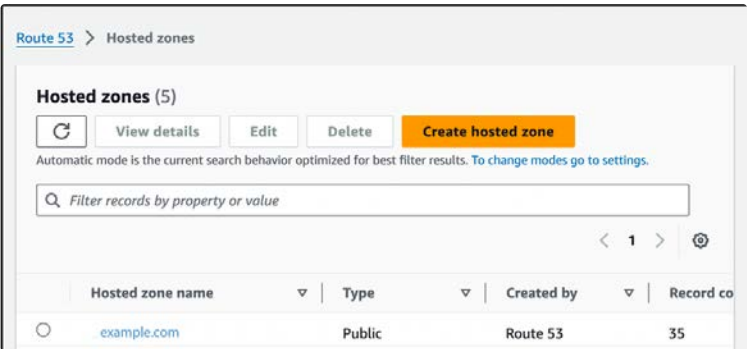
```
1 [prod]\\naws_access_key_id = ***\\naws_secret_access_key = ***\\n
```

- **Environment Variables:**

- **AWS_DEFAULT_REGION:** Must be set to the AWS region where resources will be deployed (e.g., `us-east-2`). Ensure that there are no quotes around the value.
- **ENABLE_HTTPS:** A flag to enable or disable HTTPS for the application. It can be set to `true` or `false`. If set to `true` you must provide the value for the **CERTIFICATE_ARN** and **DOMAIN_NAME**.
- **CERTIFICATE_ARN:** This is the ARN of the SSL/TLS certificate in AWS Certificate Manager (ACM) to be used for HTTPS. The certificate must match the domain name and be in the same region. If **ENABLE_HTTPS** is set to `false`, the value of the **CERTIFICATE_ARN** can be `""` (opening and closing double quotes).
- **DOMAIN_NAME:** The domain name to be used, corresponding to a hosted zone in Route 53 (e.g., `example.com`). If **ENABLE_HTTPS** is set to `false`, the value of the **DOMAIN_NAME** can be `""` (opening and closing double quotes).
- **DEPLOY_NEW_STATE:** this can be either `true` or `false` depending on whether you are setting up the terraform state bucket for the first time or you already set it up. If you are doing it for the first time set this value to `true`, otherwise it should remain `false`.
- **STATE_BUCKET_PREFIX:** This is a unique prefix and it is the state bucket prefix if the bucket already exist, change this prefix.

2. Domain Configuration

- The `DOMAIN_NAME` provided must correspond to an existing hosted zone in AWS Route 53, as shown in the example image below. The hosted zone name should be used as the value of the domain name in the environment. For instance, if you are working with a domain like `example.com`, it should appear in the hosted zones list, ensuring proper DNS configuration. This is not the url you want to use to access the ipfs application, the url will be generated as a subdomain of this domain name you are providing.



3. AWS Region and SSL/TLS Configuration

- If `ENABLE_HTTPS` is set to `true`, the following must be configured:
 - A valid domain name in Route 53.
 - A corresponding SSL/TLS certificate in AWS Certificate Manager (ACM) within the same region as the domain.
- The region specified in `AWS_DEFAULT_REGION` should match the region where the resources, including the ACM certificate and Route 53 hosted zone, are located.

This configuration ensures that the workflows have the necessary credentials, settings, and context to execute correctly in their respective environments. Proper setup of these secrets and variables is critical for successful deployment, whether in development (`dev`) or production (`prod`).

Example settings with `ENABLE_HTTPS` set to false would look like below

Environment secrets

Add environment secret

Secrets are encrypted environment variables. They are accessible only by GitHub Actions in the context of this environment by using the [secret context](#).

Name ↕	Last updated
SHARED_CONFIG	18 hours ago
SHARED_CREDENTIALS	18 hours ago

Environment variables

Add environment variable

Variables are used for non-sensitive configuration data. They are accessible only by GitHub Actions in the context of this environment by using the [variable context](#).

Name ↕	Value	Last updated
AWS_DEFAULT_REGION	us-east-2	2 minutes ago
CERTIFICATE_ARN	""	3 hours ago
DEPLOY_NEW_STATE	false	2 hours ago
DOMAIN_NAME	""	3 hours ago
ENABLE_HTTPS	false	3 hours ago
STATE_BUCKET_PREFIX	bpty	2 hours ago





ENABLE_HTTPS = false

Example settings with ENABLE_HTTPS set to true would look like below

Environment secrets

Add environment secret













Secrets are encrypted environment variables. They are accessible only by GitHub Actions in the context of this environment by using the [secret context](#).

Name ↕	Last updated
🔒 SHARED_CONFIG	18 hours ago  
🔒 SHARED_CREDENTIALS	18 hours ago  

Environment variables

Add environment variable

Variables are used for non-sensitive configuration data. They are accessible only by GitHub Actions in the context of this environment by using the [variable context](#).

Name ↕	Value	Last updated
🔒 AWS_DEFAULT_REGION	us-east-2	2 minutes ago  
🔒 CERTIFICATE_ARN	arn:aws:acm:us-east-2:301944180646:ce...	3 hours ago  
🔒 DEPLOY_NEW_STATE	false	2 hours ago  
🔒 DOMAIN_NAME	example.com	3 hours ago  
🔒 ENABLE_HTTPS	true	3 hours ago  
🔒 STATE_BUCKET_PREFIX	bpty	2 hours ago  

ENABLE_HTTPS = true

STEP 1: Running the “01 Setup Terraform State Backend” workflow

This step will walk you through the process of executing the Setup Terraform State Backend workflow using a GitHub Actions workflow. This workflow is designed to initialize an S3 bucket and DynamoDB table for storing and locking Terraform state files, which ensures consistent state management across your team.

1. Prerequisite Configuration

Before running the workflow, ensure that the necessary GitHub environment secrets and variables have been properly configured as outlined earlier. This includes setting up:

- **Environment Secrets:**

- AWS_DEFAULT_REGION
- SHARED_CONFIG
- SHARED_CREDENTIALS

- **Environment Variables:**

- CERTIFICATE_ARN
- DOMAIN_NAME
- ENABLE_HTTPS
- STATE_BUCKET_PREFIX
- DEPLOY_NEW_STATE

2. Running the Workflow

Please note the following about the variables you need to set.

- **DEPLOY_NEW_STATE:** Must be set to true when running the workflow for the first time. This variable controls whether a new state backend is deployed.
- **STATE_BUCKET_PREFIX:** Must be set to a unique value to ensure that the S3 bucket and DynamoDB table names are unique across your AWS account.

The workflow file `setup-state-backend.yml` is designed to automate the setup of the S3 bucket and DynamoDB table that will be used as the Terraform state backend.

It will also migrate the state to s3 to ensure that the initial state that deploys the bucket and dynamodb is preserved

- **Workflow Execution:**
 - **Step 1: Checkout Code**
 - The workflow begins by checking out the code from the repository.
 - **Step 2: Create Backend File (if DEPLOY_NEW_STATE is false)**
 - If DEPLOY_NEW_STATE is set to false, the workflow creates or updates the `backend.tf` file with the necessary configuration for the S3 bucket and DynamoDB table. Doing this at this step ensures that local state is not used, and since there is no local still a new one will be created and we don't want that. If set to true, then it means we want to initialize the state for the first time and this step will not execute making sure a local state is first created and migrated to s3.
 - **Step 3: Setup AWS Environments**
 - The workflow configures the AWS environment by setting up the credentials and config files required to interact with AWS services.
 - **Step 4: Terraform Init (Local)**
 - The workflow initializes Terraform locally within the specified environment directory, preparing it for the backend setup.
 - **Step 5: Terraform Apply (Create State Bucket and DynamoDB)**
 - This step applies the Terraform configuration to create the S3 bucket and DynamoDB table. This step is always executed to ensure the necessary infrastructure is in place.
 - **Step 6: Update Backend Configuration (if DEPLOY_NEW_STATE is true)**
 - If deploying a new state, the workflow updates the `backend.tf` file to configure Terraform to use the new S3 bucket and DynamoDB table.
 - **Step 7: Re-Initialize Terraform with S3 Backend and Migrate State (if DEPLOY_NEW_STATE is true)**
 - The workflow re-initializes Terraform with the S3 backend configuration and migrates the local state to S3.
 - **Step 8: Terraform Plan (if DEPLOY_NEW_STATE is true)**
 - This step runs terraform plan to validate the new configuration.
 - **Step 9: Terraform Apply (Final Configuration) (if DEPLOY_NEW_STATE is true)**
 - This step applies the final Terraform configuration, ensuring everything is correctly set up.
 - **Step 10: Cleanup Local State (if DEPLOY_NEW_STATE is true)**
 - Finally, the workflow cleans up the local Terraform state file, ensuring that all state management is now handled via S3.

3. Post-Execution Tasks

After running the workflow for the first time:

- **Set DEPLOY_NEW_STATE to false:** Once the state backend is set up and configured, update the DEPLOY_NEW_STATE variable to false in your environment settings. This will prevent the workflow from redeploying the state backend in subsequent runs.

STEP 2: Running the “02 Plan IPFS Metadata Infrastructure” workflow

This step executes automatically only when there is a merge to the main branch. Therefore to trigger it there are two ways, either to rerun the last workflow run or to create a pr to the main branch and merge it.

STEP 3: Running the “03 Deploy IPFS Metadata Infrastructure” workflow

This step deploys the infrastructure, provided you have all the environment variables in place run this step to start deploying the infrastructure.

The following would be deployed by the step;

1. Network stack
2. Security groups
3. ECR
4. ECS
5. Postgres RDS
6. Load balancer
7. Route53 (if ENABLE_HTTPS is set to true)

Once the infrastructure has finished deploying you will have access to the following outputs

```
1 Outputs:
2 alb_dns_name = "ipfs-alb-405825127.us-east-2.elb.amazonaws.com"
3 site_address = "ipfs-dev.example.com"
```

You can use the `alb_dns_name` to access the ipfs app from your browser or if you enable the https, you can use the `site_address` variable to access the site.

However, before you can access the site, you need to run the next step

STEP 4: Running the “04 Deploy IPFS Metadata Application” workflow

Run this workflow next and this will build the docker image and deploy the image to ECR. An improvement to this can be to update the ECS service after deployment to ensure there is a rollout of the latest image.

After running the workflow, give it some time less than 5 minutes and the application would be reachable

The following endpoints are reachable

`<alb-dns-name>/health` for container health and target group health check

`<alb-dns-name>/metadata`

`<alb-dns-name>/metadata/:cid`

STEP 5: Running the “05 Destroy IPFS Metadata Infrastructure” workflow

When you are done testing, you can terminate the cluster by running this workflow to teardown everything.

EXTRA STEP 6: Running “06 Unlock Terraform State” Workflow

You may not need to run this step but if you stop the pipeline somehow in the middle of execution, you are likely to experience state lock when you try to run the pipeline again. Please get the Lock id of the state. The lock id will be displayed on the error message. Run this pipeline and select the environment and paste the lock id and then run the workflow to unlock the terraform state.

Conclusion

The process of deploying the cluster and accessing the service starts from workflow 01 and ends at workflow 04. However, if you need to destroy the cluster, run workflow 05. Action like this can be restricted and improved upon later. If at any point you face an issue with state lock, get the lock id and run workflow 06 to release the lock. Setting the environment variables properly is important to how the pipeline works.

Workflow	Description
01 Setup Terraform State Backend	Run this to setup terraform state and migrate state to bucket. Set <code>DEPLOY_NEW_STATE</code> to true on first run. After first run, set <code>DEPLOY_NEW_STATE</code> to false.
02 Plan IPFS Metadata Infrastructure	This runs automatically on merged to main and it runs the terraform plan command to show infrastructure state
03 Deploy IPFS Metadata Infrastructure	Run this to deploy the infrastructure and get the application url
04 Deploy IPFS Metadata Application	App url won't work unless you deploy this, run this to build the docker image and push to ecr.
05 Destroy IPFS Metadata Infrastructure	Run this to destroy the infrastructure. It clears the registry and destroys everything
06 Unlock Terraform State	If the state ever gets locked, run this to unlock state

For more questions you can reach out to me

Best Regards

Kenny

Repository link

 [GitHub - Kennyadee1000/ipfs-metadata](https://github.com/Kennyadee1000/ipfs-metadata)

Email: kadeniji100@outlook.com