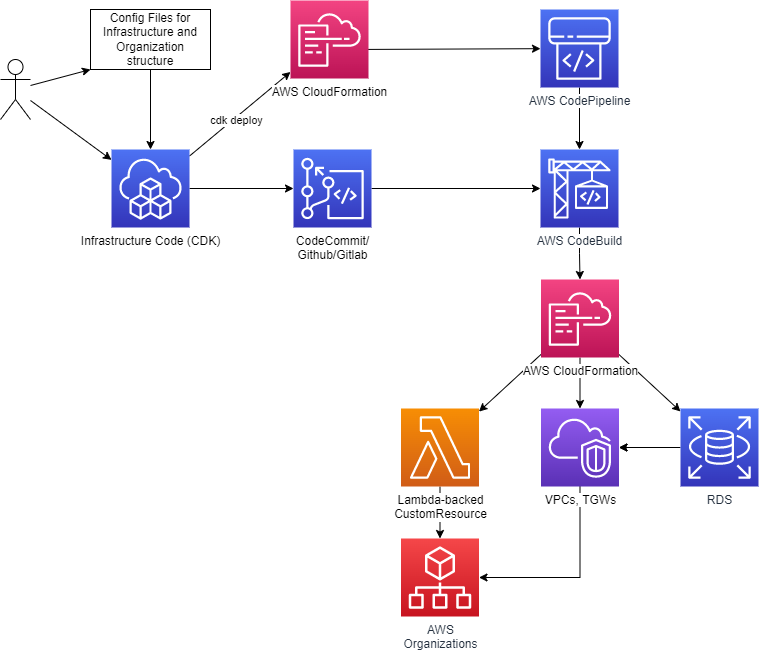
**Networking Pipeline using AWS CDK**

A common task for ProServe engagements is to deploy a landing zone for new projects. This can be a time-consuming process that contains instructions and steps other consultants have already followed before. This project utilizes the AWS CDK to provide a generic, easily configurable Infrastructure as Code (IaC) template. Using customizable configuration files, this project automates the process of setting up a networking pipeline that can provide the basics of an AWS cloud environment to help enable consultants in the deployment of landing zones. Further infrastructure can then be defined as needed by modifying the forked code.

**Architecture Diagram:**



**Solution Overview:**

The following AWS tools/services are spun up by CloudFormation for this project:

* CodePipeline
* CodeBuild
* Lambda
* AWS Organizations
* VPC, TGW, Subnets
* RDS

**Deploying the Project:**

* Fork and clone the GitHub repository.
* Initialize and bootstrap CDK for our main AWS admin account.
* Customize inputs for **pipeline**, **AWS Organizations**, and **networking**. (instructions below)
* Push CDK code to targeted repository.
* cdk deploy

**Customizing the Project:**

Pipeline**:**

This project has two files related to the deployment of CI/CD pipelines: cdk-iac-app.ts and pipeline-stack.ts.

Inpipeline-stack.ts*,* change the CodePipelineSource connection to your targeted repository.

In cdk-iac-app.ts, create a PipelineStack for each pipeline you would need, specifying an environment and branch for each pipeline created.

AWS Organizations**:**

In our CDK code, we use a Lambda-backed Custom Resource in order to use the AWS SDK and create organizational units, add/create AWS accounts to our organization, and to move those accounts to an organizational unit. Using a configuration file (*aws\_config/organization.json*), you can specify your organization structure under **OUConfig** and accounts under **AccountsConfig**. If any required parameters are missing, errors can be found logged in the CloudWatch logs of the lambda function.

Config file:

* + - Example:



* + - Organizational Units (**OUConfig**):
      * OrganizationRootId: the ID of your root organization.
        + Can be found on the AWS Organization page of the AWS console.
      * OrganizationalUnits: OUs to be created.
        + Each OU must have “Name” and “ParentName” filled.
        + List any parent OUs before their children.
    - Account Creation and Accounts Structure (**AccountsConfig**):
      * Accounts: accounts to be created/added to organization.
        + Each account must have “Email” and “Name” filled.
        + If “OrganizationalUnit” is missing, the account will be added to the root organization.

Networking:

After creating the organization structure, currently this project will require a manual step at this point. We have to create IAM credentials for our networking and workload accounts, then save those credentials as profiles in *.aws/credentials*. Then bootstrap those accounts with the following CLI commands (replace the bolded sections as described):

*cdk bootstrap* ***account-id****/****region*** *--no-bootstrap-customer-key –cloudformation execution-policies 'arn:aws:iam::aws:policy/AdministratorAccess' --trust* ***main-aws-account-id*** *--trust-for-lookup* ***main-aws-account-id*** *–profile* ***profile-name***

Replace hardcoded account IDs in env props.

A deployment will then spin up a TGW from the network account and share it with the workload accounts. The deployment will then fail due to our TGW id being hardcoded. Once that happens, simply update the hardcoded id with our just spun-up TGW and deploy/commit again.

**Code Deep Dive:**

The deploy stage consists of the following stacks:

* + OrgActivitiesStack (org-activities-stack.ts):
    - Creates a lambda function using the code in ./lambda/create-org.js
    - Configures and attaches OrganizationsIAM policies/roles for the lambda.
    - Creates a Provider and CustomResource to trigger the lambda on create or change, and get its response.
    - Reads aws\_config/organization.json and sets it as the payload our CustomResource will send to our Provider.
  + NetworkingStack (networking-stack.ts):
  + BasicVpcStack (basic-vpc-stack.ts):

**Cleanup:**

To clean up, delete the entire stack through the AWS CloudFormation console, or in the CLI we can use the CDK command “cdk destroy”

**Notes:**

* Features that are missing/could be implemented:
  + Batch script could automate initial cdk set up and enablement of aws organizations resource sharing
    - “**aws ram enable-sharing-with-aws-organization”**
  + In Organizations:
    - OU deletion
    - Moving accounts already added
      * Seems to not be worth implementing: have to find and input current OU into config file manually. Might as well manually move from console.
      * Can also tree traverse OUs and call describeOU() on each OU to find accounts, but could potentially take a lot of API calls.
    - Account deletion
      * Could do: listAccounts() to get accountIds -> check against config file -> if not in config -> closeAccount(accountId)
      * Prerequisites to be able to delete an account [(needs own billing, etc)](https://docs.aws.amazon.com/organizations/latest/userguide/orgs_manage_accounts_remove.html#orgs_manage_account-before-remove)
      * [only 10% of active accounts can be closed in 30 days](https://docs.aws.amazon.com/organizations/latest/APIReference/API_CloseAccount.html)
  + Deploy a containerized app
  + Use aurora instead of rds
  + Separate Central Networking account into a networking and a shared services account