Operations Guide

California Public Utilities Commission

RPS Database

June 2025 v1.00

This Operations Guide provides a CPUC Renewables Portfolio Standard (RPS) Database developer with information describing how the solution components and systems are assembled, updated, tested, and deployed. It includes sections on source code management, developer process, build/deploy pipeline, and security settings.

California Public Utilities Commission

Contents

CPUC RPSD System Architecture 1
Introduction 1
High-level System Architecture Diagram 1
CPUC RPS Database Architecture and Design 2

Source Code Management 6

Developer Process 0
Local Environments 0
Code Duplication Across Environments 1
CodeCommit Pull Request Details 3

Build/Deploy Pipeline 1
The Dev Pipeline 1
The Production Pipeline 6

Security 1
Access to the CPUC AWS Environment 1
User Administration 1
Security Settings 3

Appendix A - Build/Deploy Services - Terraform A-1

Appendix B - Sample Buildspec.yml B-1

CPUC RPSD System Architecture

INTRODUCTION

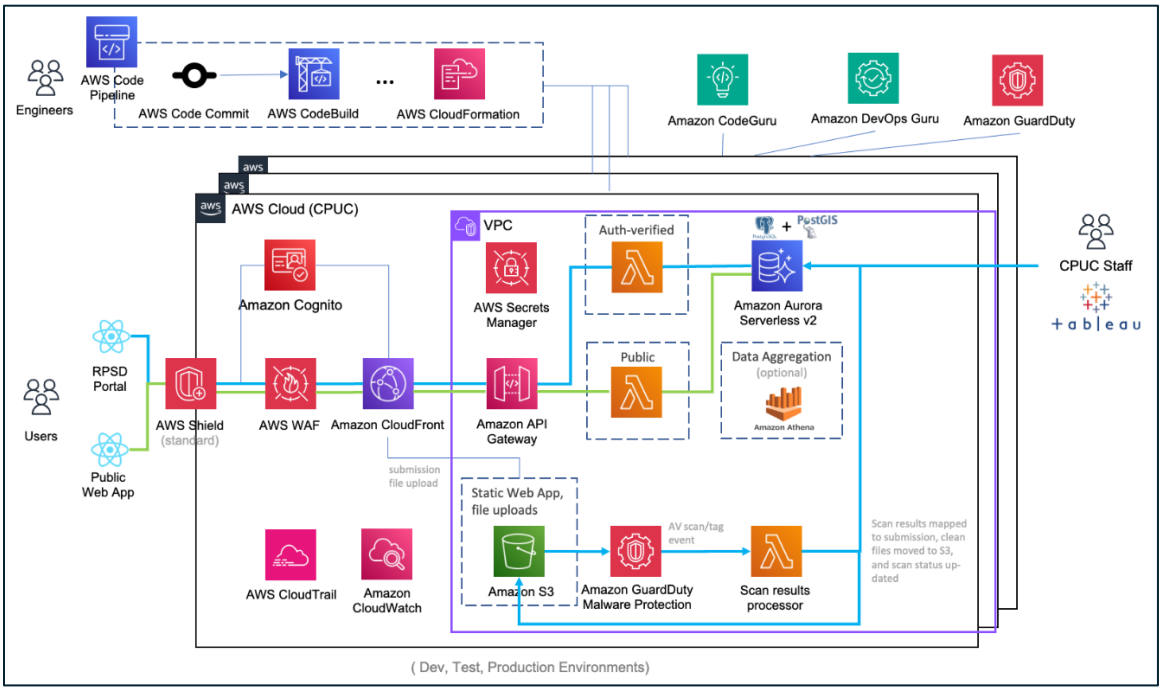
Today, many cloud providers offer services representing the latest industry standards, technological knowledge, and development best practices rolled into cloud-based service offerings. Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP) are a few examples of these.

Whether it is an IT infrastructure, security service, build and deployment pipeline, database system, data analytics, artificial intelligence system, or any of the hundreds of other service offerings, consumers of these services reap the benefits of collective knowledge, industry best practices, outsourcing of these services to experts, and the continuous improvement/innovation as these companies strive to keep up with the competition. With Amazon Web Services (AWS), modern, serverless, cloud-native applications and architecture have become a powerful, productive, and cost-effective reality for business.

HIGH-LEVEL SYSTEM ARCHITECTURE DIAGRAM

This section describes the architecturally significant parts of the design, including its resources, subsystems, packages, interfaces, and integrations. The high-level CPUC RPS Database System Architecture Diagram provided below shows the networks, boundaries and the interconnectivity between them, and the three major system areas that combine to deliver the integrated solution:

- Engineering/AWS Code Pipeline
- CPUC/AWS Cloud
- CPUC/AWS VPC



CPUC RPS DATABASE ARCHITECTURE AND DESIGN

The first thing you might notice is that there are a lot more "things" in a modern application architecture (shown in the High-level System Architecture Diagram above). Well, that is by design.

There is a fundamental principle in software design called Separation of Concerns (SoC). Instead of having one fallible, monolithic piece of software that composes your system, you have smaller components, each responsible for a specific function.

Some of the benefits of this approach are:

- Individual components can be changed, fixed, redeployed, or replaced without negatively impacting the running system. No unnecessary re-deploys!
- Each component has defined inputs and outputs, making it possible to combine them in different ways to accomplish business or system goals.
- With well-defined components, each with specific inputs, outputs, and behaviors, testing, operation, maintenance, monitoring, and issue resolutions become much easier.

Cloud providers like AWS have the ability to automatically provision IT infrastructure for you. Combined with this, AWS can deploy components and services into an infrastructure, configure them, secure them, monitor them, automatically scale to meet customer demand, derive insights, and many other capabilities. This comprehensive approach to modern architecture offers a powerful, yet cost-effective solution that enables organizations to focus on business rather than managing low-level infrastructure and services.

Let's dig into the details to further explore the benefits of AWS modern architecture:

Organizational Agility, Speed and Quality

Modern architecture, also known as serverless or event-driven, benefits from cloud provider investment in handling infrastructure complexities for you. This, as mentioned earlier, enables teams to focus on business goals, processes, applications, and other business needs.

Here are a few ways AWS enables organizational agility and speed:

- Rather than spending time building, configuring, securing, debugging, and scaling the infrastructure, architects, and engineers use tooling and code to instruct AWS to build infrastructure for them:
  - The team specifies the desired infrastructure via visual tool or descriptive source code, and AWS builds the infrastructure.
- With the time savings gained by not managing infrastructure actively, the team can focus on assembling and connecting cloud services to accomplish business process flows and application experiences.
- Once deployed, automated build and deploy services make subsequent changes a no-effort, fast, and reliable process.

Security

Security standards and best practices are implemented for you. Each AWS service offering has security built in. Here are the key areas of AWS security that are enabled by default:

Infrastructure (https://aws.amazon.com/products/security/?nc2=h_ql_prod_se). AWS provides several security capabilities and services to increase privacy and control network access. These include:

- Network firewalls built into Amazon VPC let you create private networks and control access to your applications.
- Encryption in transit with TLS, Customer controlled across AWS services.
- Connectivity options that enable private, or dedicated, connections from your office or on-premises environment.
- DDoS mitigation technologies that apply at layer 3 or 4 as well as layer 7. These can be applied as part of application and content delivery strategies.
- Automatic encryption of all traffic is enabled on the AWS global and regional networks between AWS-secured facilities.

Cloud resources can further be locked down at a very granular, policy-based approach with Identity and Access Management (IAM)

Data Encryption (https://aws.amazon.com/iam/identity-center/). AWS adds a layer of security to your cloud data at rest. This provides scalable, efficient encryption features (typically AES 256). These include:

- **Data at rest encryption** is available in most AWS services, such as Amazon EBS, Amazon S3, Amazon RDS, Amazon Redshift, Amazon ElastiCache, AWS Lambda, and Amazon SageMaker.
- **Flexible key management options**, including AWS Key Management Service, enable you to keep complete control over your encryption keys, or you can choose to have AWS manage the keys for you.
- **Dedicated, hardware-based cryptographic key storage** using AWS CloudHSM helps to satisfy your compliance requirements.
- **Encrypted message queues** for the transmission of sensitive data using server-side encryption (SSE) for Amazon SQS supports and maintains data security.

In addition, AWS provides APIs to integrate encryption and data protection with any of the services you develop or deploy in an AWS environment.

Service and Applications

In addition to data being secured at rest, data, state, and identity are secured on the wire via HTTPS or encrypted WebSocket protocols. Access to each service is controlled by an IAM security policy, with the least-privilege access enabled by default. Another interesting point: AWS automatically handles security updates, upgrades, patches, etc. for the infrastructure behind the services used. This is like having your own DevOps/SecOps team keeping your environment safe.

User Authentication and Authorization

Amazon Cognito (https://aws.amazon.com/cognito/details/) is a developer-centric, cost-effective customer identity and access management (CIAM) service. It provides a secure identity store and federation options that can scale to millions of users. Amazon Cognito supports login with social identity providers and SAML or OIDC-based identity providers for delightful customer experiences and offers advanced security features to protect your customers and business. It supports various compliance standards, operates on open identity standards (OAuth2.0, SAML 2.0, and OpenID Connect), and integrates with an extended ecosystem of front-end and back-end development resources and SDK libraries.

Identity Management

Here are a few of the major features offered by Amazon Cognito:

- **Self-Registration**: customers self-manage onboarding, two-factor authentication (2FA), profiles, and custom attributes, which reduces support issues.
- **Identity store** (user pools): an API-based user repository used to manage user profiles, custom attributes, group management, and more.
- **User authentication**: a risk-score-based user identity verification system that supports pools of users, multiple authentication sources (e.g., federated, OAuth, social), device validation, multifactor authentication (MFA), and advanced detection of issues like unusual activity, new location logins, new devices logins, compromised credentials, etc.
- **Federation**: enables users to log in via social identity providers, such as Apple, Facebook, Google, and Amazon, as well as enterprise identity providers via SAML and OIDC.
- **Access control**:
    - Secures the last mile of integration with an application. Amazon Application Load Balancers (ALBs) and API gateways have built-in policy enforcement points that provide access based on Amazon Cognito tokens and scopes.
    - Users can be dynamically mapped to different roles to support least-privilege access to services and resources like RDS, S3 buckets, etc.
    - Provides machine-to-machine security via the OAuth credential flow, which includes user token information, to ensure access is secured through the entire application and service call chain.
- **Additional security**: Cognito aligns with multiple security and compliance requirements, including those for highly regulated organizations (e.g., healthcare companies and merchants). It is HIPAA eligible and PCI DSS, SOC, and ISO/IEC 27001, ISO/IEC 27017, ISO/IEC 27018, and ISO 9001 compliant.

Integrations

AWS contains over 200 AWS service integrations, as well as thousands of community app/lib integrations, that simplify building best-practice, cloud-native applications. Much of the infrastructure and platform management efforts of the past are outsourced to the AWS IT infrastructure allowing teams to focus on creating solutions for customers and businesses. With AWS integrations, teams simply assemble and connect needed services.

Scale/High Availability

Another major advantage of the AWS cloud-native approach is auto-scale. Unlike architectures where you are actively configuring, managing, and scaling the infrastructure, with the cloud-native approach you pay for the services you use and not more.

AWS services will automatically scale up to meet increased customer demand, or scale down to save costs.

Fault tolerance

AWS services have fault tolerance built in (https://aws.amazon.com/blogs/compute/building-well-architected-serverless-applications).

Utilizing advanced monitoring, in some cases AI, when issues are detected or service failures occur, AWS will automatically attempt to retry, reroute, or reprovision requests to services located across availability zones. Here are a few examples:

- With S3 failures, AWS will attempt to retry, then request across availability zones.
- Lambda failures are similar, with the addition of timeout retries, and monitoring events.
- Aurora Serverless V2 cluster failure will:
    a. Promote another cluster, then
    b. Attempt to reprovision.

AWS also provides control to the developer/engineer to manage faults within code, handling scenarios like timeouts, code errors, service failures, etc.

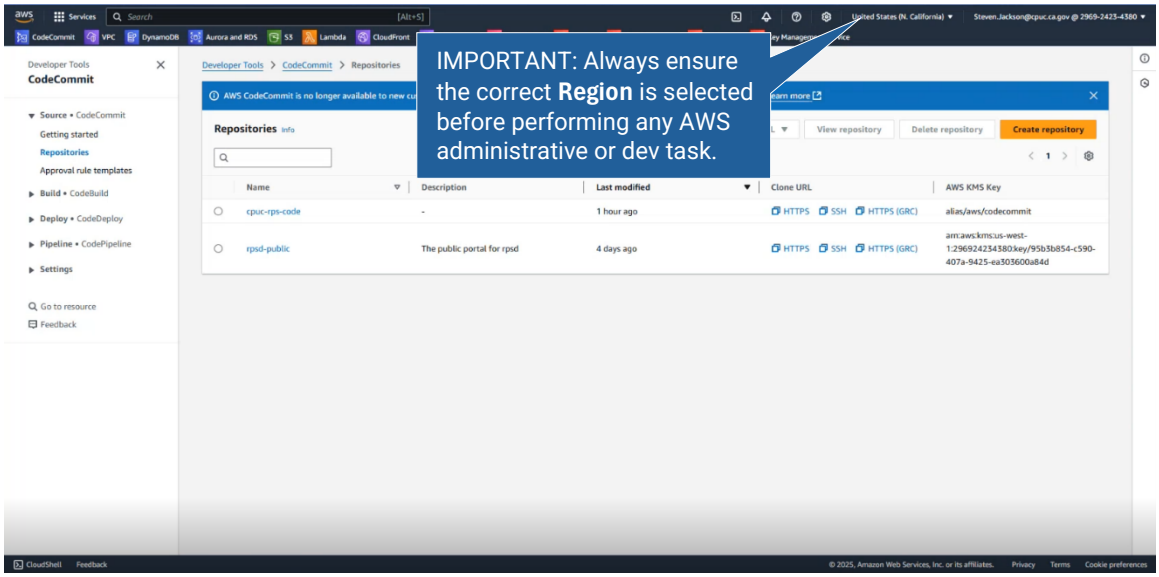
Efficiency/Cost

This feature is one of the more powerful capabilities of the AWS offering - you pay only for what you use. With cloud-native, serverless architecture, services load, unload, scale up, and scale down automatically based on user or system demand. When not in use, the service scales to zero.

With the CPUC business cycle being periodic, this aspect of AWS will provide direct benefits over the allocated infrastructure in use today.

Source Code Management

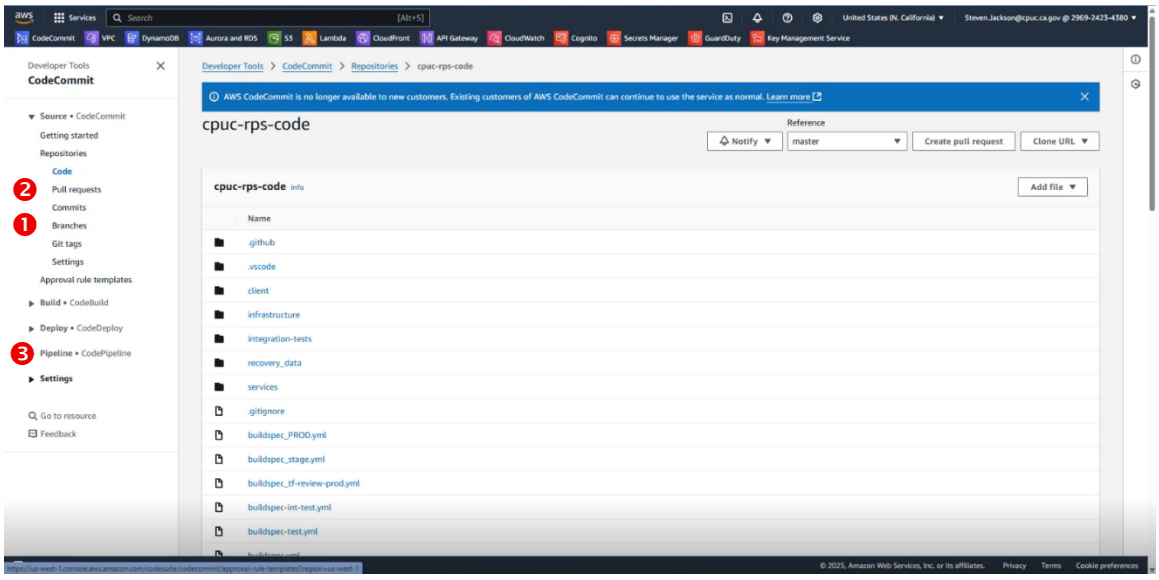
Source codes for the CPUC RPSD application and public sites are currently managed using AWS CodeCommit.



Note: the system implementation will be finalized using the CPUC GitHub enterprise repo.

CodeCommit manages the code within each repository:

2. All new code is checked into a branch.
3. Then committed to a PR and merged from the branch into the main application.
4. Once merged into main, the pipeline is initiated.



Developer Process

LOCAL ENVIRONMENTS

The dev process for initiating code development and troubleshooting is based on coding created in each dev's local environment. It is supported with an individual dev's local variables file.

This can be described as a pre-dev environment.

Note: although referred to as a local environment, all actions and commands are executed within the AWS cloud.

As an example, a new dev may need perform a troubleshooting task. This individual would proceed by accessing their environment and testing code locally in their pre-dev environment prior to moving the updated code to the Build/Deploy Pipeline described in the next section.

Once the local environment is initialized, the dev follows the same steps as contained in the pipeline. So, how does a dev test locally?

- Initialize a Terraform backend:

```
PS C:\Users\dsalins\OneDrive - InterVision\Projects\cpuc\cpuc-rps-code> cd .\infrastructure\
PS C:\Users\dsalins\OneDrive - InterVision\Projects\cpuc\cpuc-rps-code\infrastructure> terraform init "-backend-
config=backend.hcl"
```

Results (in terminal):

```
Initializing the backend...
Initializing modules...
Initializing provider plugins...
- Reusing previous version of hashicorp/null from the dependency lock file
- Reusing previous version of hashicorp/http from the dependency lock file
- Reusing previous version of hashicorp/random from the dependency lock file
- Reusing previous version of hashicorp/archive from the dependency lock file
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/random 03.1.3
- Using previously-installed hashicorp/archive v2.2.0

Terraform has been successfully initialized!
```

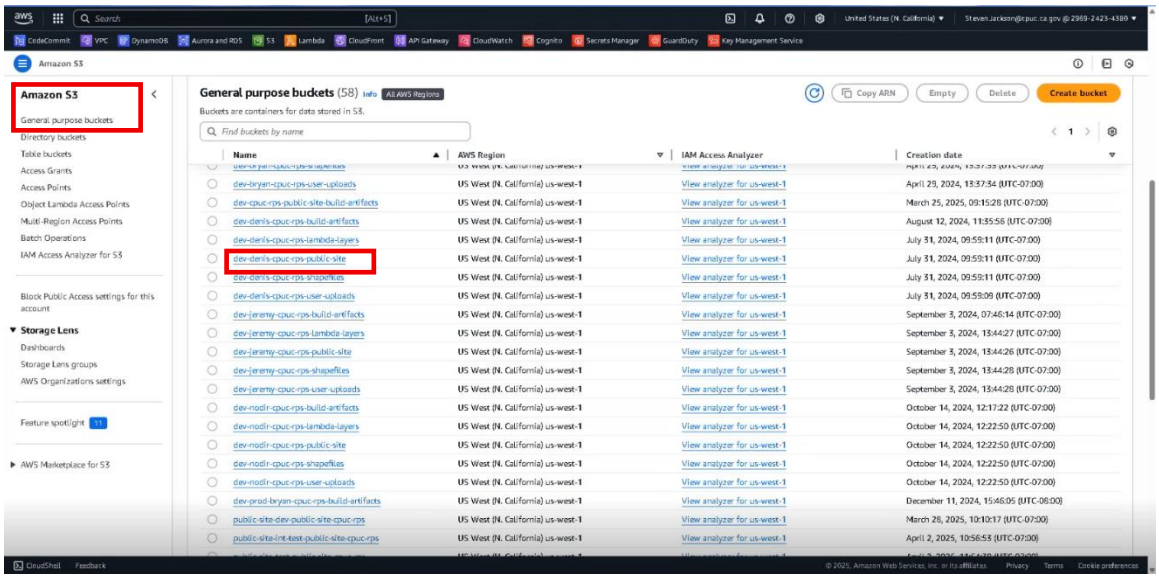
- Run a Terraform plan, specifying the target module and passing it a variables file:

```
PS C:\Users\dsalins\OneDrive - InterVision\Projects\cpuc\cpuc-rps-code\infrastructure> terraform plan -
target="module.project" -var-file="vars/dev-denis-env.tfvars"
```

The plan (apply) commands to initialize the local dev environment with the local variables:

```
build:
  commands:
    - "cp ../client/portal/rps/.base.env ../client/portal/rps/.env"
    - "terraform apply -target.'module.project' -var-file'vars/rpsd-inttest-cpuc-env.tfvars' --auto-approve"
    - "terraform output >> ../client/portal/rps/.env"
    - "echo 'VITE_CPUC_ORG_ID - ef9e60ce-368f-4d1c-9af1-4ebd217df0ef' >> ../client/portal/rps/.env"
```

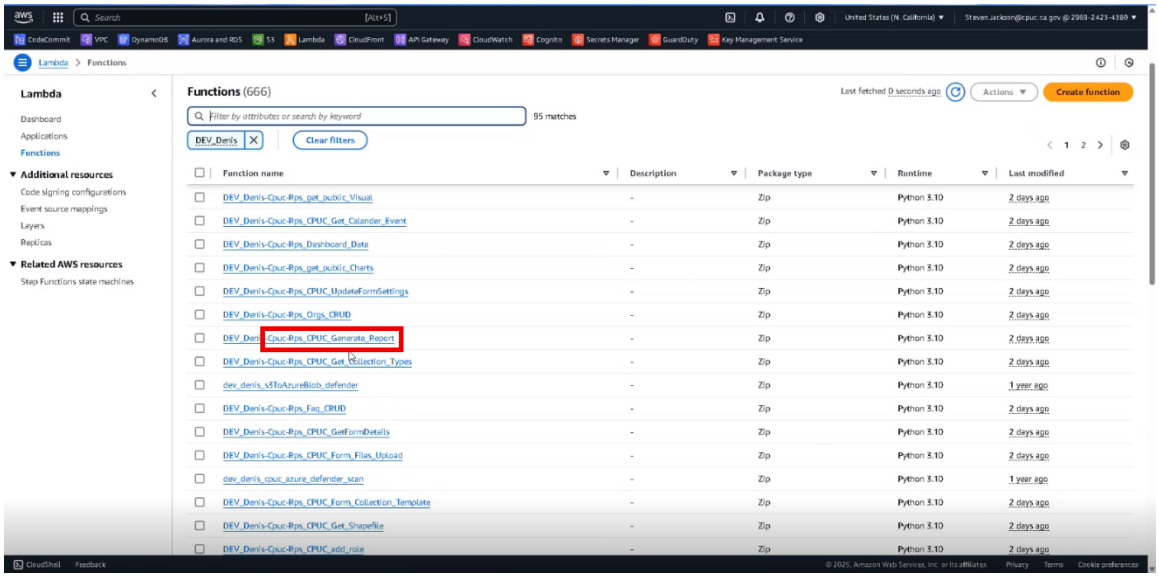
Running the Terraform plan creates a local S3 bucket for development and testing activities. This S3 bucket is the dev's isolated local environment, created for their exclusive use. It is accessed at **Amazon S3** > **General purpose buckets**. This structure allows devs to work concurrently and not interfere with each other's work.



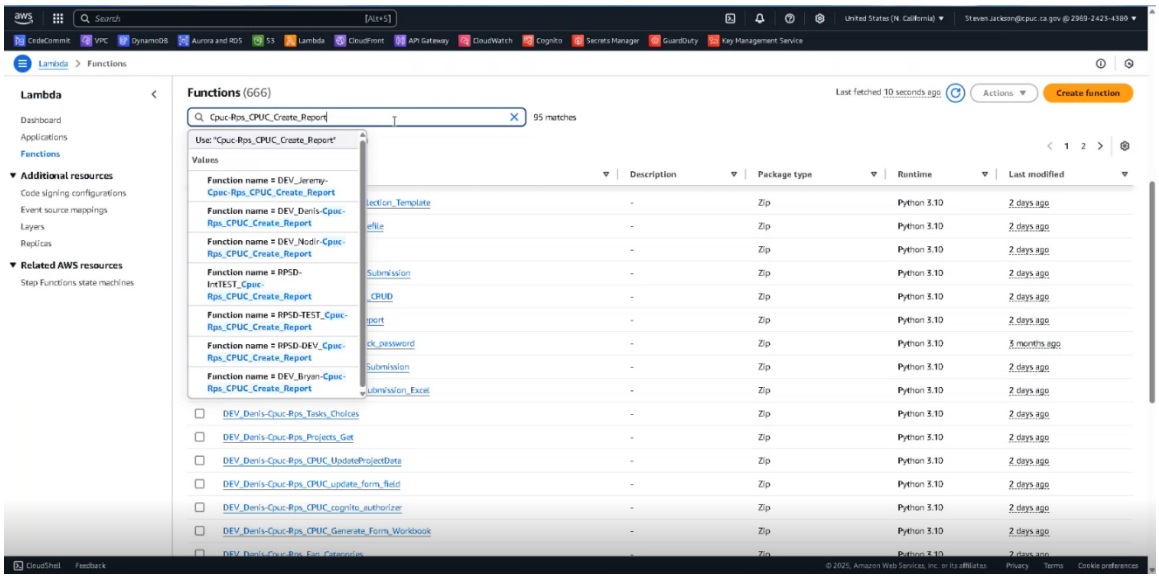
When the dev is at a point where their code is ready for system testing, they perform a PR (pull request) and merge their code into master. This flows the code branch into the build/deploy pipeline.

CODE DUPLICATION ACROSS ENVIRONMENTS

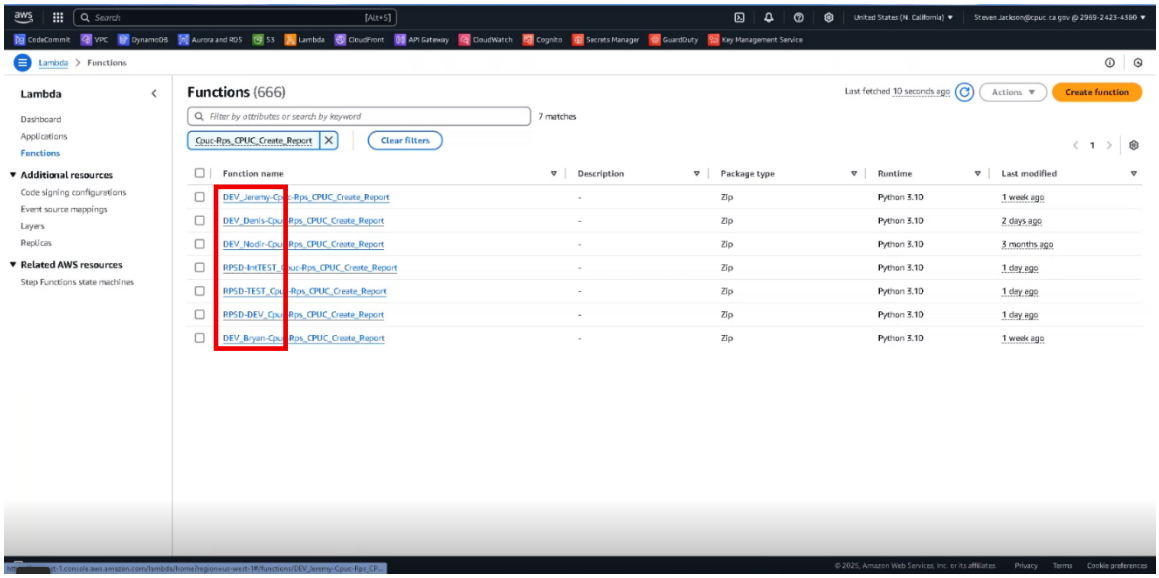
If a dev has created, or has updated, code in their local environment, AWS will automatically cascade the code across all local, dev, and prod environments. This allows individual devs to maintain their own code while still making it available for safe use from within the other environments. For example, from the list of an individual dev's Lambda **Functions** shown below, a search for **Cpuc-Rps_CPUC_Create_Report** will display that same Lambda's name except duplicated with the prefix for all available environments.



- Search string specified:

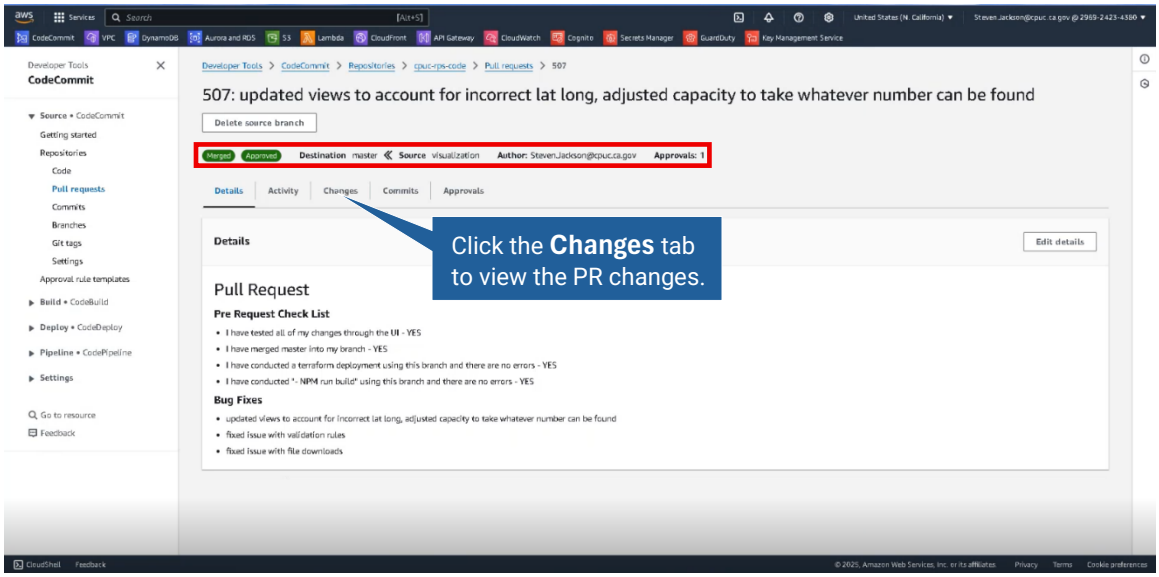


- Updated **Functions** list shows the standardized Lambda is duplicated and available in all environments:

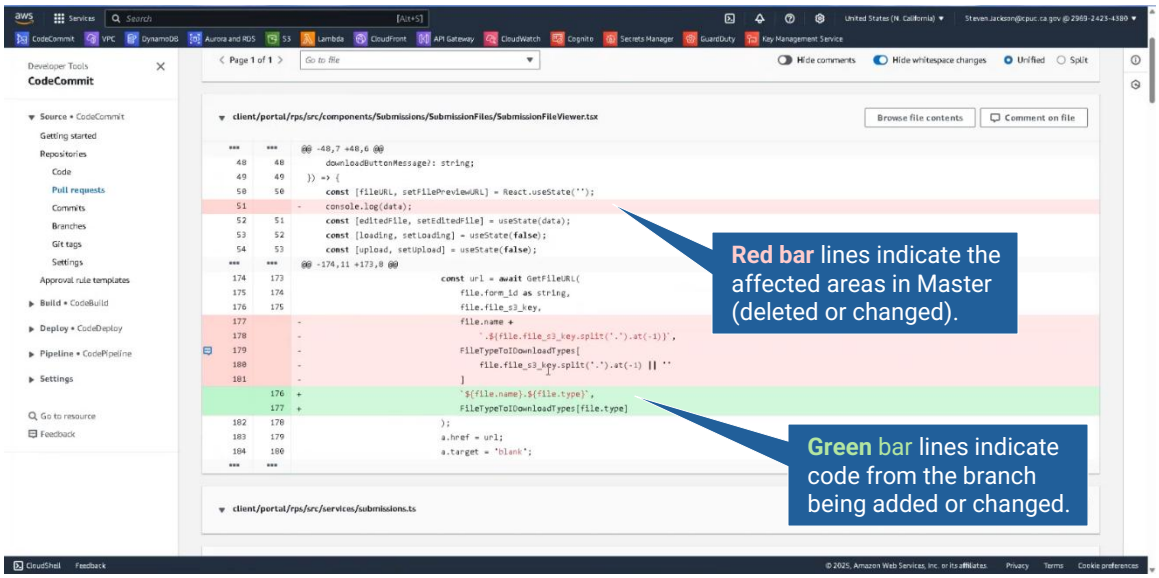


CODECOMMIT PULL REQUEST DETAILS

AWS generates a report that tracks dev activity and provides PR information identifying the status (**Merged**, **Approved**), **Source** (branch), **Destination**, **Author**, and number of **Approvals** for any PR. In this case we are seeing changes in the **visualization** branch being merged into **master**.



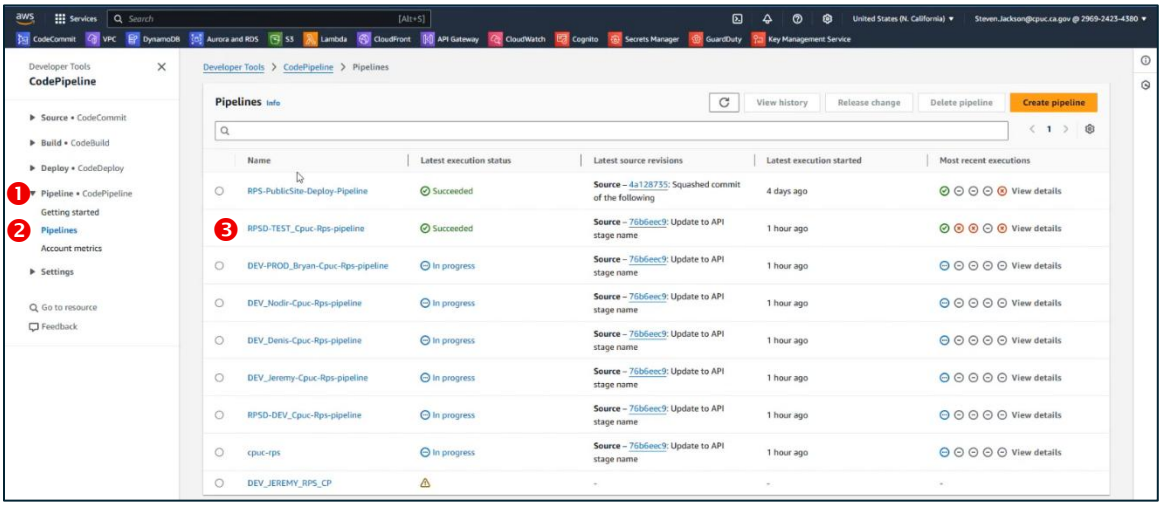
The **Changes** view is color coded to provide a clear record of what changes are being made, and which lines/areas are being updated.



This view also auto-updates line numbers to provide additional clarifying information to identify what specifically is being updated. In the example we see line 51 is deleted and line 52 and beyond are shifted up by one line. We also see lines 177 - 181 are being deleted and replaced by the new lines 176 and 177 and the remaining code beyond that point is pulled up and renumbered accordingly.

Build/Deploy Pipeline

The system is updated via automated pipelines. A pipeline comprises steps~~, each~~ controlled by a ~~specific~~ YAML buildfile (see Appendix B - Sample Buildspec.yml). This file contains commands needed to access variables, invoke commands, and integrate code to point where it can be tested, reviewed, and approved. To view a pipeline, ❶click **Pipeline • CodePipeline** , ❷select **Pipelines** to view all pipelines, and ❸click on a pipeline to open.



THE DEV PIPELINE

The following are the account and location details for both CPUC sites' **development pipelines**:

- **RPSD-TEST_Cpuc-Rps-pipeline** (DEVELOPMENT) (AWS: 296924234380)
  - **Development Portal site**:
    - p1rpsd.cpuc.ca.gov (rpsd-dev)
    - rpsd-int-test does not have a public friendly url
    - p2rpsd.cpuc.ca.gov (rpsd-test)
- **RPS-PublicSite-Deploy-Pipeline** (DEVELOPMENT) (AWS: 296924234380)
  - **Development Public site**:
    - p1rpsdata.cpuc.ca.gov (rps-publicsite-dev)
    - p2rpsdata.cpuc.ca.gov (rps-publicsite-test)

The pipelines (above) have the following deployment stages for each pipeline:

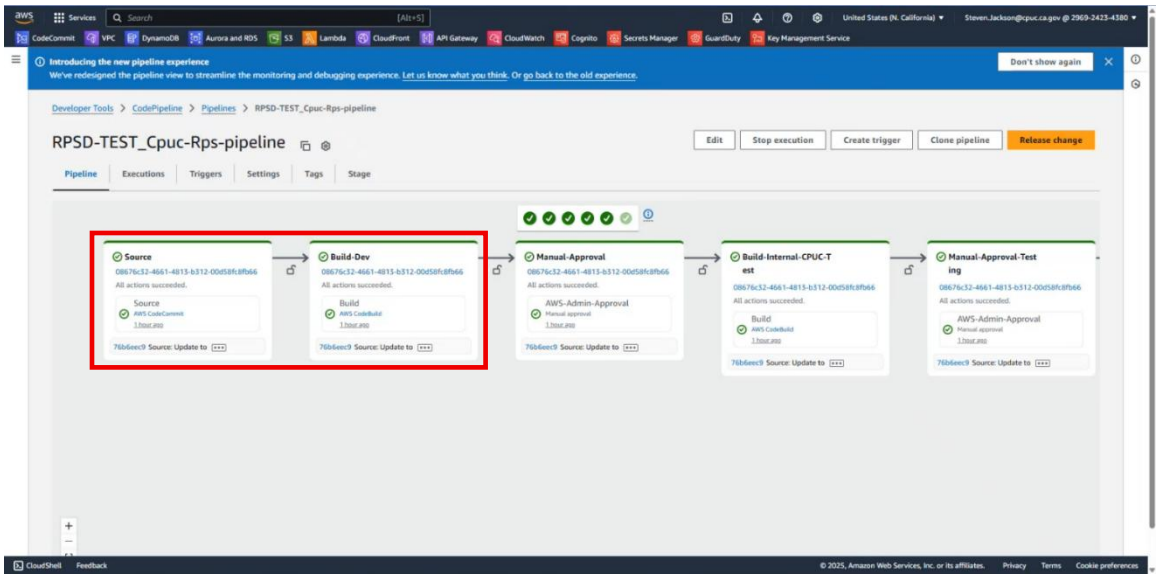
- LOCAL-DEV (DEV-Denis, Dev-Bryan, Dev-Jeremy, Dev-####) performs a GIT PR to Merge into Master Branch (see above 0 CodeCommit Pull Request Details).
  - Dev - "rpsd-dev-"
  - Manual Approval
  - Int Test - "rpsd-int-test-"
  - Manual Approval

- Test - "rpsd-test-"

The following are the steps within each (above) pipeline deployment stage:

- Download Tools:
  - Get Terraform
  - Install Liquibase (automation)
  - Install PowerShell (execution)
  - Manual Backup of DB
    - "aws rds create-db-cluster-snapshot"
    - Excluded in rpsd-dev stage.
  - Install React NPM Tool Packages for site
  - Deploys Terraform (build, and post-build commands)
- Deploy IAC Backend
- Deploy React Frontend Web Site
  - NPM Run Build

As stated in the previous section, when a PR is issued, the system is alerted to a new branch "XYZ" ready to go into master. XYZ has changes and will require team review/approval. Once the PR is confirmed, XYZ merges into master. This action initiates pipeline automation. When updated code is checked in, the change activates the pipeline **Source** module. The pipeline then activates **Build-Dev** and begins integrating the code.

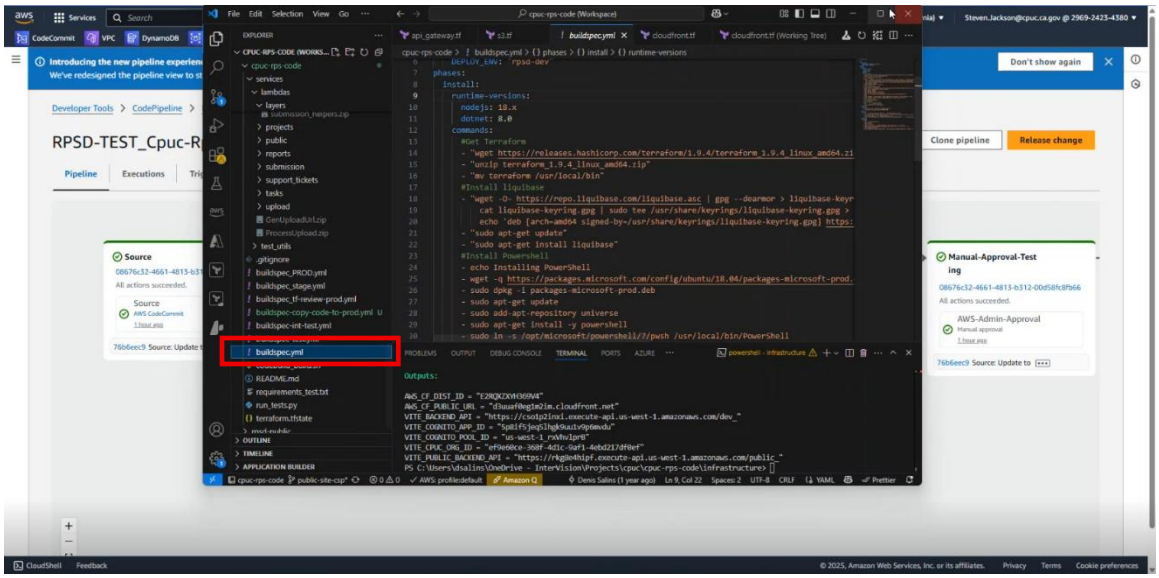


The **Build-Dev** module attempts to create the new build using parameters specified in the buildspec.yml file. Essential actions from the buildspec file are the same here as are used in the dev environment:

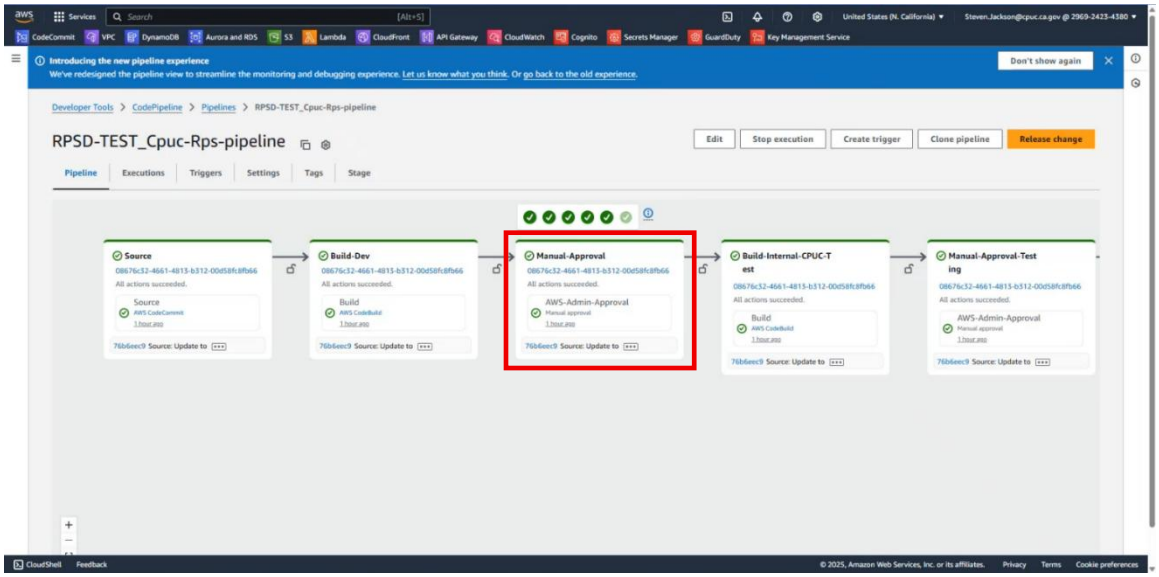
- Get Terraform
- Install Liquibase (automation)
- Install PowerShell (execution)
- Build the Node Package Manager (NPM)

- Deploys Terraform (pre-build, build, and post-build commands)

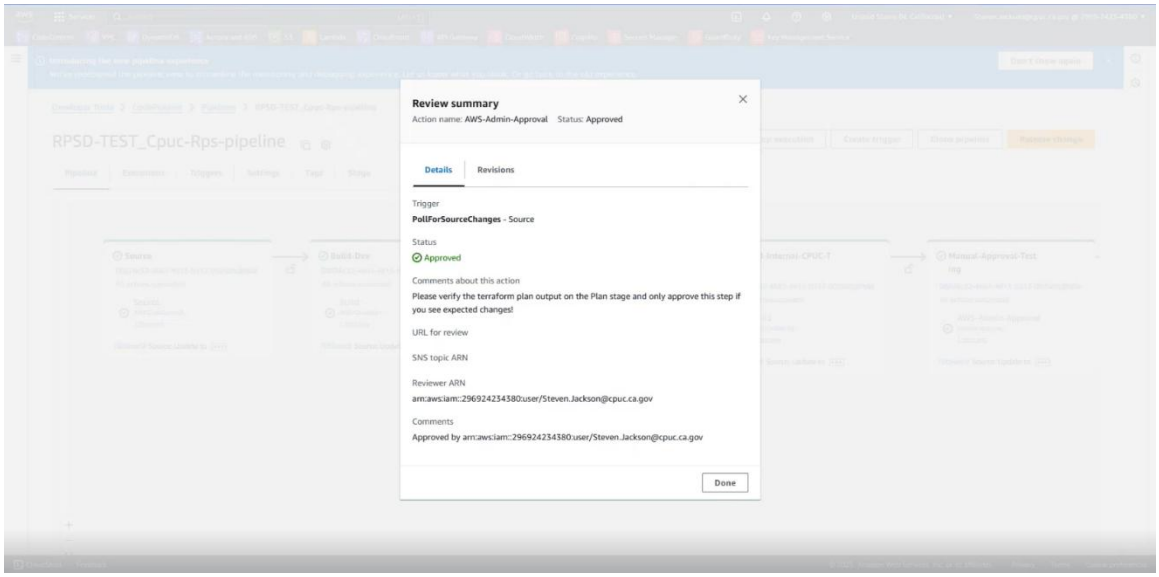
Once the Terraform build is done, the NPM broadens and installs any new components, executes the run build, and creates a CloudFront invalidation that clears the cache for CloudFront and delivers the new site.



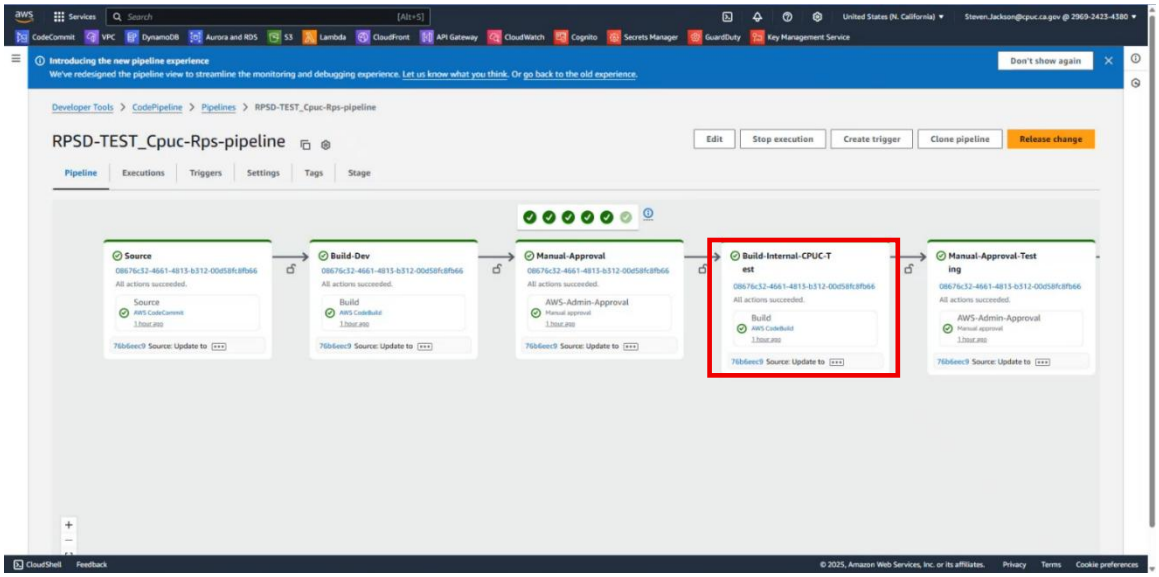
Once the **BuildDev** is deployed, the site is up and running with the new code. It then requires a manual approval process.



A team member must access the new code review and approval process via the pipeline interface to confirm the new code meets their approval.

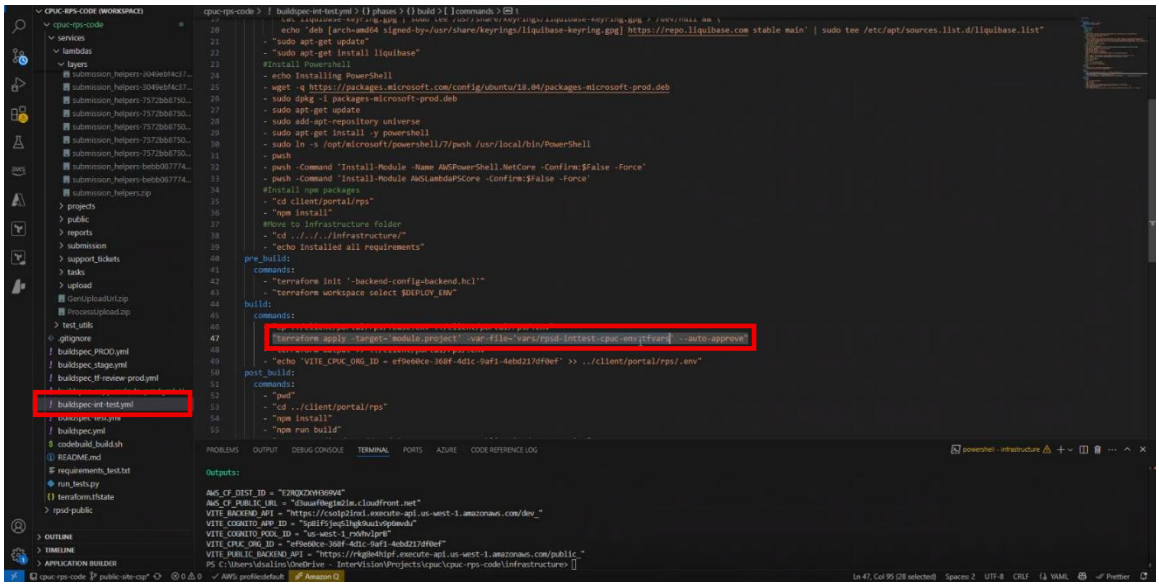


After **Manual-Approval** is confirmed, the pipeline progresses to the next stage - **Build-Internal-CPUC-Test**.

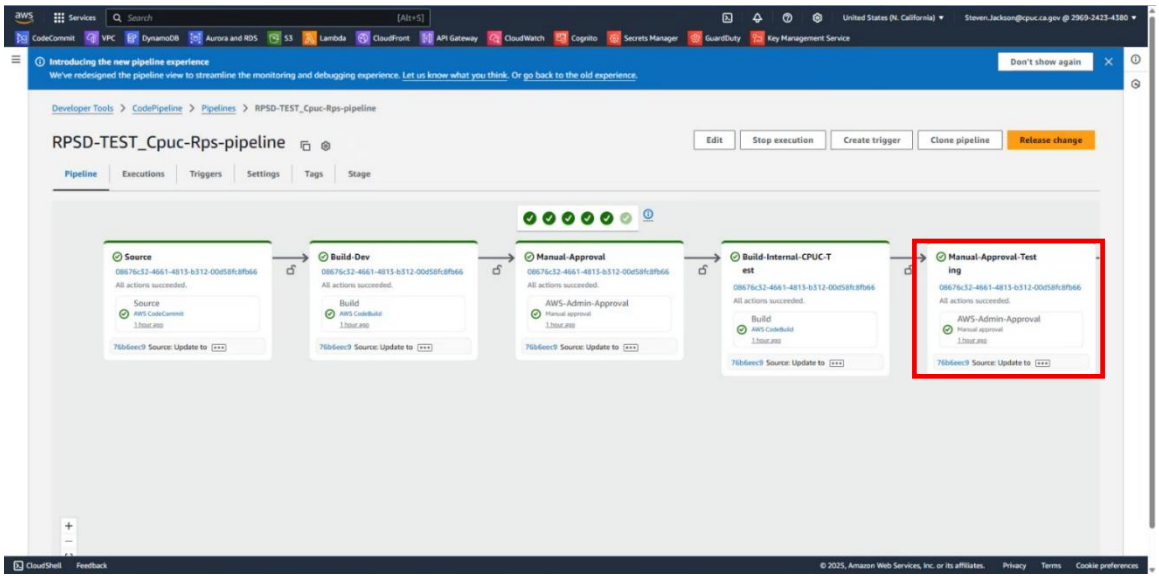


Build-Internal-CPUC-Test is now controlled via the buildspec-int-test.yml file and is usually configured very similarly to the previous **Build-Dev** test. An exception would be this code references a different variables file for the test.

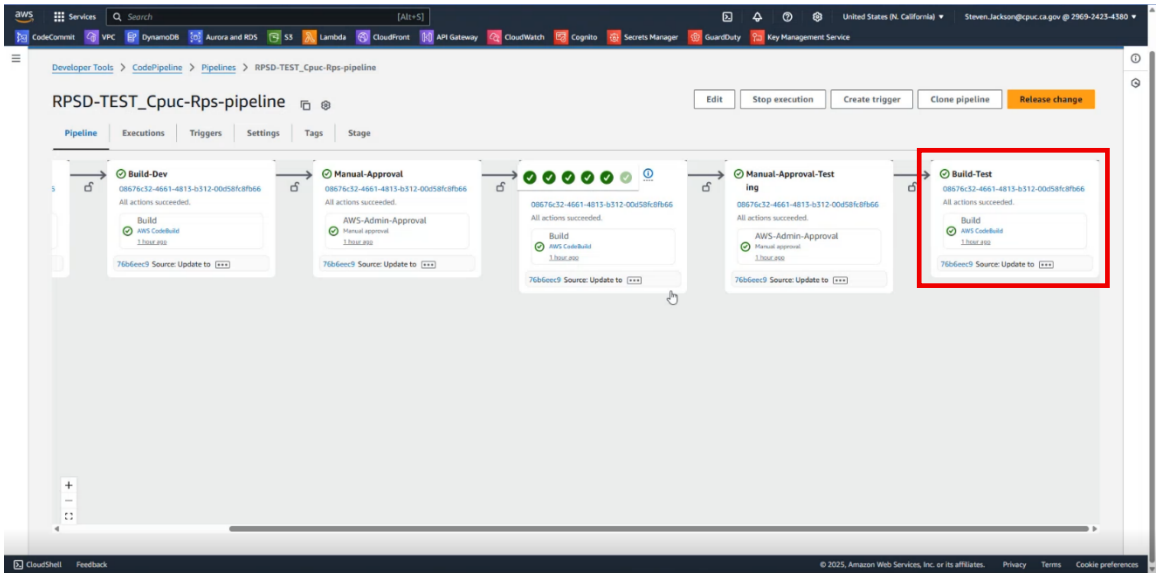
As in the earlier test, essential actions once the Terraform build is done, the NPM broadens and installs any new components, executes the run build, and creates a CloudFront invalidation that clears the cache for CloudFront and delivers the new site.



After the invalidation is created, the pipeline progresses to the next pipeline stage - **Manual-Approval-Testing**. The resulting build will again require manual approval from a team member.



This approval confirmation initiates the next pipeline element - **Build-Test** and there is another build process. At this point, the site is up and running in the dev environment.



THE PRODUCTION PIPELINE

Pipeline Locations

The following are the account and location details for both CPUC sites' **production pipelines**:

- **RPSD-PROD_Cpuc-Rps-pipeline** (PRODUCTION) (AWS: 025066257111)
  - **Production Database site**:
    - p3rpsd.cpuc.ca.gov (rpsd-stage)
    - rpsd.cpuc.ca.gov (rpsd-prod)
- **RPS-PublicSite-Deploy-Pipeline** (PRODUCTION) (AWS: 025066257111)
  - **Production Public site**:
    - p3rpsdata.cpuc.ca.gov (rps-publicsite-stage)
    - rpsdata.cpuc.ca.gov (rps-publicsite-prod)

Pipeline Deployment Stages

Each pipeline above has the following deployment stages:

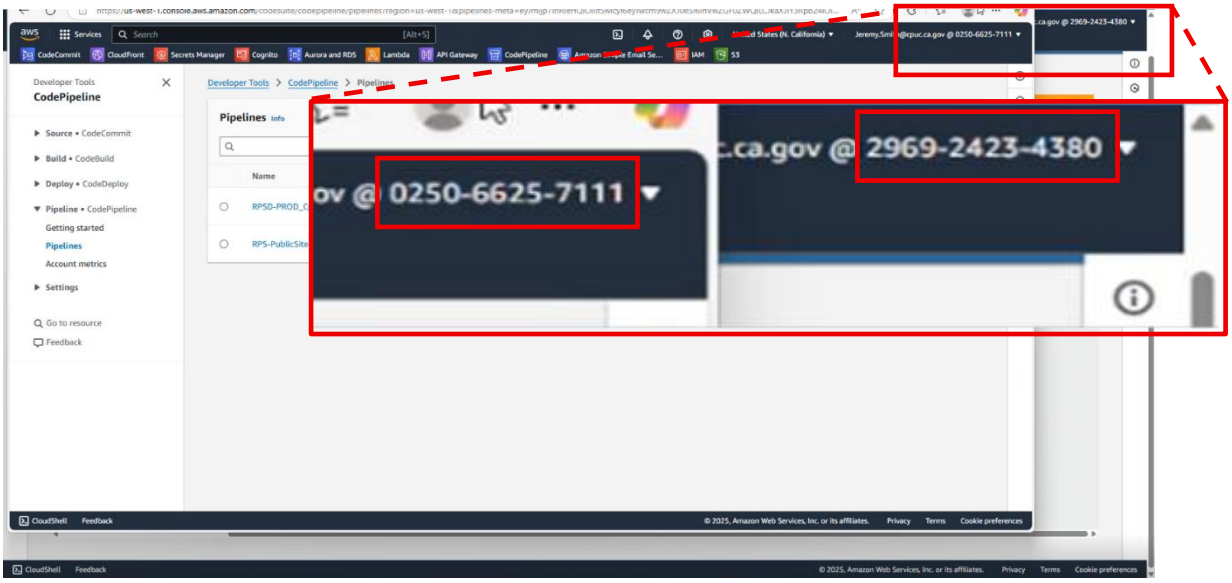
- Stage - "rpsd-stage-"
- Manual Approval
- PLAN Prod - Final Manual Review Step
- Manual Review and Approval for Prod deploy
- Prod - "rpsd-prod-"

Each pipeline deployment stage contains the following steps:

- Download Tools:
  - Get Terraform
  - Install Liquibase (automation)
  - Install PowerShell (execution)
  - Manual Backup of DB
    - "aws rds create-db-cluster-snapshot"
      - Install React NPM Tool Packages for site
      - Deploys Terraform (build, and post-build commands)
  - Deploy IAC Backend
  - Deploy React Frontend Web Site
    - NPM Run Build

Dev Environment to the Stage / Production Environment

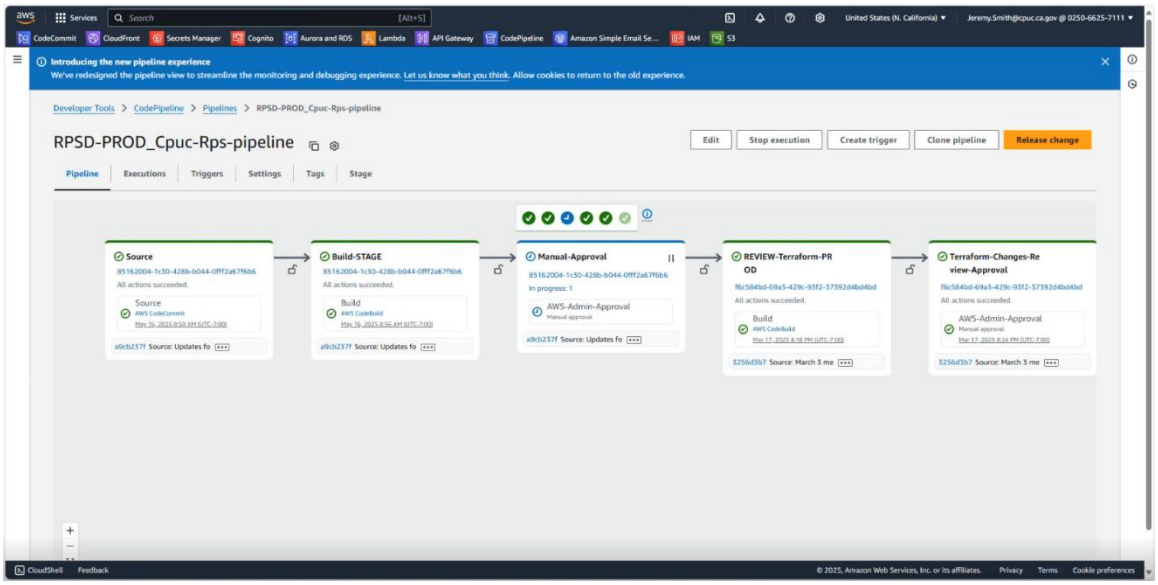
The code now must be moved from the dev environment to the stage / production environment - a different AWS account. The pipeline and infrastructure are all the same in both environments.



Once moved to stage / production, the source code changes and that environment's pipeline becomes active. The pipeline functions the same as the dev pipeline.

Note: a manual review process is recommended in **REVIEW-Terraform-PROD** to confirm and document the changes that are going to be made in Terraform.

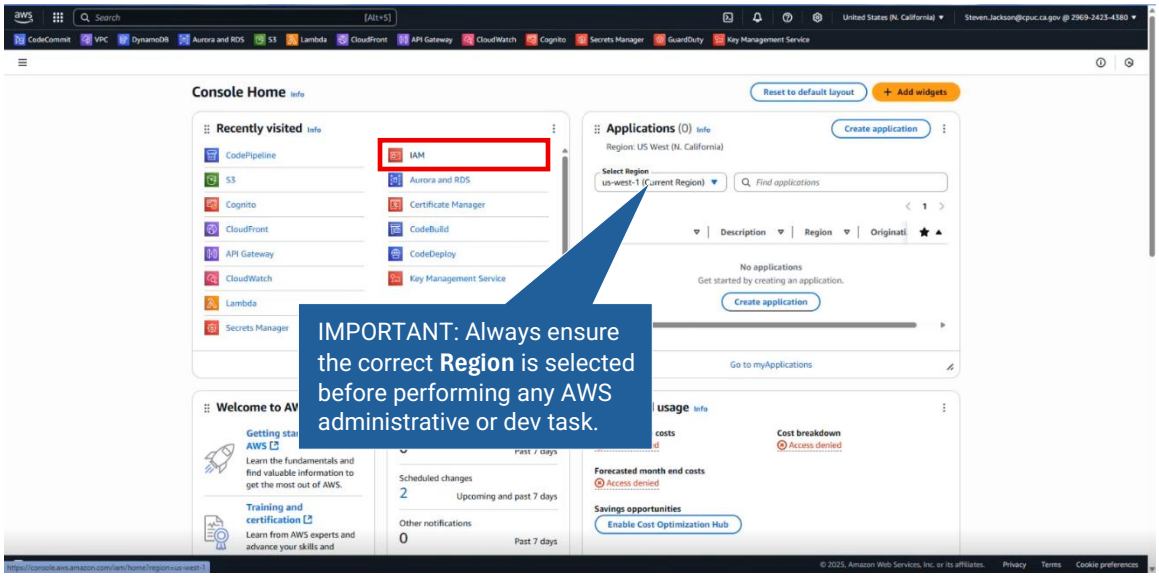
Then the pipeline pushes the new code out to production, and it is deployed.



Security

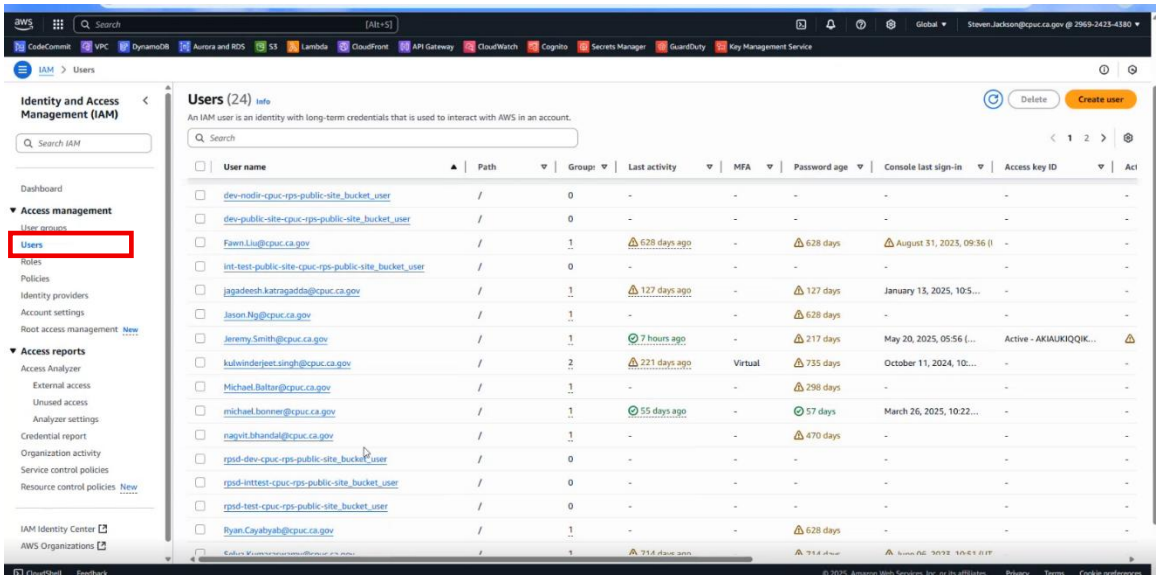
ACCESS TO THE CPUC AWS ENVIRONMENT

CPUC RPSD IT group admins create/update user profiles. This activity is performed on the AWS Console Home screen. Users are added and permissions are updated via the IAM (Identity and Access Management) page. AWS IAM is a web service that allows you to securely control access to AWS resources, manage users, groups, and roles, and define policies these entities can perform on AWS resources.



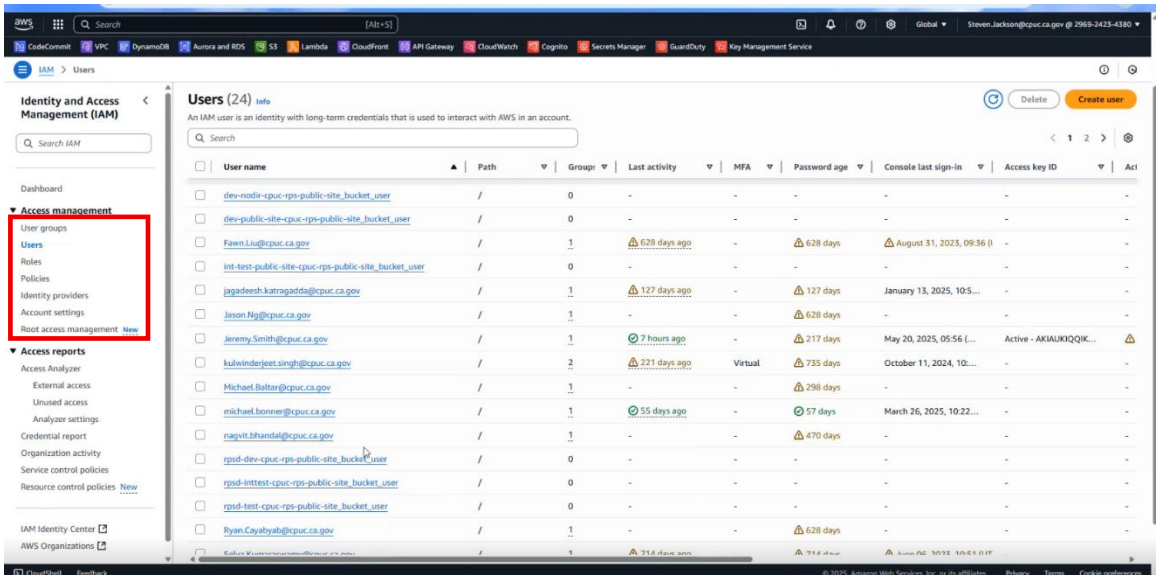
USER ADMINISTRATION

Access users from the **Access Management** > **Users** menu.



The other security/access management related administrative functions available from this page are:

- User groups: collections of users that are granted the same set of permissions and access to resources. Defining a User Group allow you to grant or revoke permissions to multiple users at once, rather than individually.
- Roles: IAM roles are entities that grant permissions to access AWS resources without requiring users to have long-term credentials like access keys.
- **Policies**: In AWS, a policy defines permissions, specifying what actions can be performed on which AWS resources. Policies are used to control access to AWS resources and ensure only authorized users or applications can access sensitive data. Policies are essential for maintaining security and compliance within the AWS environment.
- **Identity providers**: systems that handle user authentication and identity management, allowing users to access AWS resources through their existing credentials.
- **Account settings**: AWS account settings encompass various configurations that define how your AWS account operates. These settings cover aspects like account management, security, billing, access control, etc.
- **Root access management**: AWS root access management is a feature within AWS Organizations that allows central control and management of root user credentials and actions across member accounts.



SECURITY SETTINGS

Database Encryption

```
  storage encrypted = true
```

Database encryption is enabled.

```
vpc_security_group_ids = ["${aws_security_group.db_instance_securityGroup.id}"]

security_group_rules = {
  vpc_ingress = {
    cidr_blocks = module.vpc.private_subnets_cidr_blocks
  }
```

Database security group is defined to only allow certain ports access.

S3 Storage

```
rule {
  apply_server_side_encryption_by_default {
    sse_algorithm = "AES256"
  }
}
```

S3 Storage is encrypted using AES256.

API Gateway/Lambda

```
resource "aws_api_gateway_gateway_response" "cors_unauthorized" {
   rest_api_id   = aws_api_gateway_rest_api.cpuc_rps_api.id
   status_code   = "401"
   response_type = "UNAUTHORIZED"

   response_parameters = {
      "gatewayresponse.header.Access-Control-Allow-Origin" = "${var.allowLocalhost ? "'*'" : var.useCloudfrontURL ? "'https://${aws_cloudfront_distribution.s3_distribution.domain_name}'" : "'$
      {var.formalDomain}'"}",
   }
}
```

Secure access to the API gateway and Lambda's via the CloudFront domains.

```
resource "aws_lambda_permission" "user_review_task_allow_api" {
  statement_id  = "AllowAPIgatewayInvokation"
  action        = "lambda:InvokeFunction"
  function name = aws_lambda_function.user_review_task.function_name
  principal     = "apigateway.amazonaws.com"
}
```

AWS Lambda's are only accessible via authorized API gateway invocation.

```
module "submissions_forms_formId_workbook_cors" {
   source  = "squidfunk/api-gateway-enable-cors/aws"
   version = "0.3.3"
   allow_origin  = "${var.allowLocalhost ? "*" : var.useCloudfrontURL ? "https://${aws_cloudfront_distribution.s3_distribution.domain_name}" : var.formalDomain}"

   api_id          = aws_api_gateway_rest_api.cpuc_rps_api.id
   api_resource_id  = aws_api_gateway_resource.submissions_forms_formId_workbook.id
}
```

Cross-Origin resource sharing (CORS) enabled on API Gateway enabled for each Lambda to ensure security source. (https://docs.aws.amazon.com/AmazonS3/latest/userguide/cors.html).

```
resource "aws_iam_role_policy_attachment" "form_validation_get_lambda_policy" {
  role       = "${aws_iam_role.form_validation_get_lambda_exec.name}"
  policy_arn = aws_iam_policy.DB_access.arn
}

resource "aws_iam_role" "form_validation_get_lambda_exec" {
  name = "${var.enviro}_${var.project}_cpuc_form_validation_get_lambda_exec_role"
  assume_role_policy = aws_iam_role.DB_access.assume_role_policy
}
```

GuardDuty

```
resource "aws_guardduty_malware_protection_plan" "cpuc-s3-user-uploads-gdprotection" {
Click to collapse the range. .guardduty_role.arn

  protected_resource {
    s3_bucket {
      bucket_name = aws_s3_bucket.cpuc-user-uploads.id
    }
  }

  actions {
    tagging {
      status = "ENABLED"
    }
```

GuardDuty is enabled for bucket upload and will perform active virus scanning on the entire bucket and contained files.

VPC/Network

```
name = "${var.enviro}-${var.project}-vpc""
cidr = "10.99.0.0/16"

azs                     = var.vpc_azs # removed "${var.aws_region}c", for us-west-1
public_subnets          = var.public_subnets
private_subnets         = var.private_subnets
database_subnets        = var.database_subnets
default_route_table_tags = { DefaultRouteTable = true }

enable_dns_hostnames    = true
enable_dns_support      = true
enable_nat_gateway      = false # Disabled NAT to be able to run this example quicker
enable_vpn_gateway      = true
single_nat_gateway      = true
one_nat_gateway_per_az = false
```

The network and VPC are grouped into Public, Private, and Database subnets. The Private subnet has query access to the database subnet. Public is limited to only public facing points. This restricts the public facing access points to prohibit/restrict access to the private and/or database subnets.

AWS Cognito Pool

```
resource "aws_cognito_user_pool" "user_pool" {
  name = "${var.enviro}_${var.project}_User_Pool"

  auto_verified_attributes  = ["email"]
  alias_attributes          = ["preferred_username"]

  password_policy {
    minimum_length = 14
  }

  mfa_configuration = "OPTIONAL"
  sms_authentication_message = "Your security code is {####}"

  software_token_mfa_configuration {
    enabled = true
  }
```

AWS Cognito user pools provide several security benefits such as multi-factor authentication (MFA) to enhance user sign-in security by requiring a second form of verification. Additionally, it includes threat protection features that monitor sign-ins for risk indicators and can block suspicious activities. AWS Cognito also integrates with other AWS services such as AWS Lambda and API Gateway, allowing you to build secure, serverless applications. These features collectively help secure the application against network intrusion, password guessing, and user impersonation.

AWS manages security on multiple fronts, covering both infrastructure as well as network level requirements. This includes the security of the data centers, servers, and databases, as well as network segmentation, among others. Additionally, AWS manages the updating and patching responsibilities of the host operating system and the physical security of all server hardware and infrastructure.

Service Deletion Prevention

The CPUC RPSD system has the following components with Deletion Prevention enabled to ensure that data and or configuration is not lost during a modification or change to the service.

- RDS Database

```
deletion_protection = true
```

- Cognito Pool

```
lifecycle {
  prevent_destroy = true
}
```

- S3 Buckets User Uploads

```
lifecycle {
  prevent_destroy = true
}
```

AWS WAF (Web Application Firewall) and Shield Policy

Amazon services WAF and Shield are included when using AWS CloudFront. Specifically, these services enhance protection against standard threats made by random attackers on the internet. WAF and Shield provide high level protection against many typical malicious attacks such as a DDoS (distributed denial of service). Additionally, if events occur, AWS automatically logs the event and information from the event (e.g., the attacker's IP address).

WAF is regionally configured to only allow US traffic to protect the AWS API Gateway and Cognito pool.

```
resource "aws_wafv2_web_acl_logging_configuration" "waf_logging" {
  log_destination_configs  = [aws_cloudwatch_log_group.waf_log_group.arn]
  resource_arn             = aws_wafv2_web_acl.us_only_waf.arn
}

resource "aws_wafregional_geo_match_set" "geo_match_set" {
  name = "${var.enviro}_${var.project}_WAF_geo_match_set"

  geo_match constraint {
    type = "Country"
    value = "US"
  }
}

resource "aws_wafv2_web_acl_association" "api_gateway_pri_waf_association" {
  resource_arn = aws_api_gateway_stage.api_gateway_stage.arn
  web_acl_arn = aws_wafv2_web_acl.us_only_waf.arn
}

resource "awswafv2_web_acl_association" "cognito_pool_waf_association" {
  resource_arn = aws_cognito_user_pool.user_pool.arn
  web_acl_arn = aws_wafv2_web_acl.us_only_waf.arn
}
```

Key Management/Secrets Manager

AWS Key Management and Secrets Manager ensure keys/secrets are auto rotated and securely managed. Program functionality queries the key/secret as needed ensuring the latest secure key/secret is referenced.

AWS Backup

The RPSD application leverages AWS Backup plan backup functionality. The backups capture the S3 buckets and Aurora database. Two plans are implemented:

- Daily backup: daily backups are retained for 30 days.
- Monthly backup: monthly backups are retained for 12 months.

```
resource "aws_backup_selection" "daily" {
  count        = local.daily_backup_count
  iam_role_arn = aws_iam_role.service_role[0].arn
  name         = "${var.enviro}_${var.project}_CPUC_daily"
  plan_id      = aws_backup_plan.daily[0].id
  resources    = ["${module.aurora_postgresql_v2.cluster_arn}",
                  "${aws_s3_bucket.cpuc-user-uploads.arn}",
                  "${aws_s3_bucket.cpuc-shapefiles.arn}",
                  "${aws_s3_bucket.lambda_layers_bucket.arn}",
                  ]
}
```

Appendix A - Build/Deploy Services - Terraform

Requirements

- Chocolatey (For Windows as Admin) - https://chocolatey.org/install.
- Terraform - https://developer.hashicorp.com/Terraform/tutorials/aws-get-started/install-cli.
- AWS CLI - https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html.
- Liquibase (Used for Migrations) - https://docs.liquibase.com/start/install/home.html.

Install Virtual Environment

- pip3 install virtualenv

Create a Virtual Environment

- virtualenv -p python3 .venv

Start Virtual Environment

- &'{PATH TO venv}\scripts\activate

Install Terraform

- choco install Terraform
- choco install git

Setup AWS access

- AWS configure

Terraform Create Workspace

- Terraform workspace new {name}

Terraform Initialization

Open the Terraform Main Folder in the VSCode and in the root directory (that contains other directories like modules, template etc.), and type the command given below for Terraform initialization:

- Terraform init '-backend-config=backend.hcl'

The backend has shared buckets. These manage those shared buckets across the build states.

TERRAFORM APPLY

Once Terraform is initialized, run the command below to see which resources get deployed if the Terraform script runs successfully:

- To deploy the pipeline for a workspace, replace the -target argument below with "module.pipeline".
- Terraform plan -var-file="vars/{INSERT_ENV_File}.tfvars".
  - ex. Terraform plan -target="module.project" -var-file="vars/dev-cpuc-env.tfvars".

The variable configuration file is **Terraform.auto.tfvars**. You can change the attribute parameters values there to suit your needs/preferences.

To deploy infrastructure, run the command below in the root folder:

- Terraform apply -var-file="vars/{INSERT_ENV_File}.tfvars"
  - ex. Terraform apply -target="module.project" -var-file="vars/dev-cpuc-env.tfvars"

Note: remember to run above command in the /Terraform_infra_1 folder. Otherwise, it will return ERROR due to a file location issue.

After the command starts to run, Terraform will show the resources being created and will ask for your permission to build up resources.

Enter "yes" to initiate the AWS infrastructure build up and deployment.

TERRAFORM DESTROY

If you want to destroy the Terraform, type the command below:

- Terraform destroy -var-file="vars/{INSERT_ENV_File}.tfvars"
  - ex. Terraform destroy -var-file="vars/dev-cpuc-env.tfvars"

On running this, Terraform will show resources being destroyed and will ask your permission to clean up resources.

Type - **yes** - and AWS infrastructure will start the cleanup, and the resources will be destroyed.

REQUIREMENTS

No requirements.

PROVIDERS

Name	Version
archive	2.3.0
aws	4.54.0
random	3.4.3

MODULES

Name	Source	Version
db	Terraform-aws-modules/rds/aws	n/a
s3_bucket	Terraform-aws-modules/s3-bucket/aws	~> 3.4.0
vpc	Terraform-aws-modules/vpc/aws	n/a

RESOURCES

Name	Type
aws_api_gateway_integration.hello_world	resource
aws_api_gateway_method.hello_world	resource
aws_api_gateway_resource.hello_world	resource
aws_api_gateway_rest_api.hello_world	resource
aws_codepipeline.codepipeline	resource
aws_codestarconnections_connection.example	resource
aws_iam_role.codepipeline_role	resource
aws_iam_role.lambda_exec	resource
aws_iam_role_policy.codepipeline_policy	resource
aws_lambda_function.hello_world	resource
aws_s3_bucket.codepipeline_bucket	resource
aws_s3_bucket_acl.codepipeline_bucket_acl	resource
aws_security_group.db_instance_securityGroup	resource
random_string.test_suffix	resource
archive_file.hello_world	data source
aws_iam_policy_document.this	data source
aws_region.current	data source

INPUTS

Name	Description	Type	Default	Required
allocated_storage	n/a	any	n/a	yes
aws_region	n/a	any	n/a	yes
backup_window	n/a	any	n/a	yes

Name	Description	Type	Default	Required
db_name	n/a	any	n/a	yes
deletion_protection	n/a	any	n/a	yes
engine	n/a	any	n/a	yes
engine_version	n/a	any	n/a	yes
family	n/a	any	n/a	yes
identifier	n/a	any	n/a	yes
instance_class	n/a	any	n/a	yes
maintenance_window	n/a	any	n/a	yes
password	n/a	any	n/a	yes
port	n/a	any	n/a	yes
private_subnets	n/a	any	n/a	yes
public_subnets	n/a	any	n/a	yes
tags	n/a	any	n/a	yes
username	n/a	any	n/a	yes
vpc_cidr	n/a	any	n/a	yes

OUTPUTS

Name	Description
hello_world_url	n/a

Appendix B - Sample Buildspec.yml

```
version: 0.2

env:
  variables:
    AWS_DEFAULT_REGION: "us-west-1"
    DEPLOY_ENV: "rpsd-test"
phases:
  install:
    runtime-versions:
      nodejs: 18.x
      dotnet: 8.0
    commands:
      #Get Terraform
      - "wget https://releases.hashicorp.com/terraform/1.9.4/terraform_1.9.4_linux_amd64.zip"
      - "unzip terraform_1.9.4_linux_amd64.zip"
      - "mv terraform /usr/local/bin"
      #Install liquibase
      - "wget -O- https://repo.liquibase.com/liquibase.asc | gpg --dearmor > liquibase-keyring.gpg && \
          cat liquibase-keyring.gpg | sudo tee /usr/share/keyrings/liquibase-keyring.gpg > /dev/null && \
          echo 'deb [arch=amd64 signed-by=/usr/share/keyrings/liquibase-keyring.gpg] https://repo.liquibase.com stable main'
| sudo tee /etc/apt/sources.list.d/liquibase.list"
      - "sudo apt-get update"
      - "sudo apt-get install liquibase"
      #Install Powershell
      - echo Installing PowerShell
      - wget -q https://packages.microsoft.com/config/ubuntu/18.04/packages-microsoft-prod.deb
      - sudo dpkg -i packages-microsoft-prod.deb
      - sudo apt-get update
      - sudo add-apt-repository universe
      - sudo apt-get install -y powershell
      - sudo ln -s /opt/microsoft/powershell/7/pwsh /usr/local/bin/PowerShell
      - pwsh
      - pwsh -Command 'Install-Module -Name AWSPowerShell.NetCore -Confirm:$False -Force'
      - pwsh -Command 'Install-Module AWSLambdaPSCore -Confirm:$False -Force'
      ## Deletes the existing manual snapshot then creates a latest
      - aws rds delete-db-cluster-snapshot --db-cluster-snapshot-identifier rpsd-test-backup
      - aws rds create-db-cluster-snapshot --db-cluster-identifier rpsd-test-cpuc-rps-postgresqlv2 --db-cluster-snapshot-
identifier rpsd-test-backup
      #Install npm packages
      - "cd client/portal/rps"
      - "npm install"
      #Move to infrastructure folder
      - "cd ../../../infrastructure/"
      - "echo Installed all requirements"
  pre_build:
    commands:
      - "terraform init '-backend-config=backend.hcl'"
      - "terraform workspace select $DEPLOY_ENV"
  build:
    commands:
      - "cp ../client/portal/rps/.base.env ../client/portal/rps/.env"
      - "terraform apply -target='module.project' -var-file='vars/rpsd-test-cpuc-env.tfvars' --auto-approve"
      - "terraform output >> ../client/portal/rps/.env"
      - "echo 'VITE_CPUC_ORG_ID = ef9e60ce-368f-4d1c-9af1-4ebd217df0ef' >> ../client/portal/rps/.env"
  post_build:
    commands:
      - "pwd"
      - "cd ../client/portal/rps"
      - "npm install"
      - "npm run build"
      - "aws s3 cp dist/ s3://$DEPLOY_ENV-cpuc-rps-public-site/ --recursive"
      - "AWS_CF_DIST_ID=$(grep AWS_CF_DIST_ID .env | cut -d'=' -f 2- | tr -d '\"')"
      - "echo $AWS_CF_DIST_ID"
      - "aws cloudfront create-invalidation --distribution-id $AWS_CF_DIST_ID --paths '/*' --debug"
```