Logo

Description automatically generated

# Software Development Lifecycle

## Change Management Process

Change management is managed with AWS IAM Policies, AWS CodeCommit and Git.

### Developer Access

All developers are provisioned via AWS IAM account and associated to the Developer Group Role. The developer group role prohibits any direct code commits to the master branch. This ensures that the git change management process is followed at a policy level.

### Development Workflow

**1. Defining a new feature/bug:**

For a new feature or a bug, create a story in ServiceNow with the following information:

1. **"Short description"** - The Title/short description of the story
2. **"Acceptance Criteria"** - Define the acceptance/approval items needed for the ticket to pass QA successfully. Please make the acceptance clear so that it can be tested without you being present.
3. **"Points"** - Define level of effort in points. (hourly)
4. **"Assigned to"** - Owner who will be responsible for the story.
5. **"Sprint"** - Assign to the current sprint or it will not be visible in the sprint board. Leave blank to assign it to the backlog.
6. **"Status"** - Set to "Ready" (Meaning the AC is all defined and ready for a developer to pickup.). If the AC is not fully defined, leave status set to "Draft" and leave a comment as to what is needed. Note: Bugs require an additional field "Steps to reproduce" which should give detailed instructions on how to reproduce the issue.

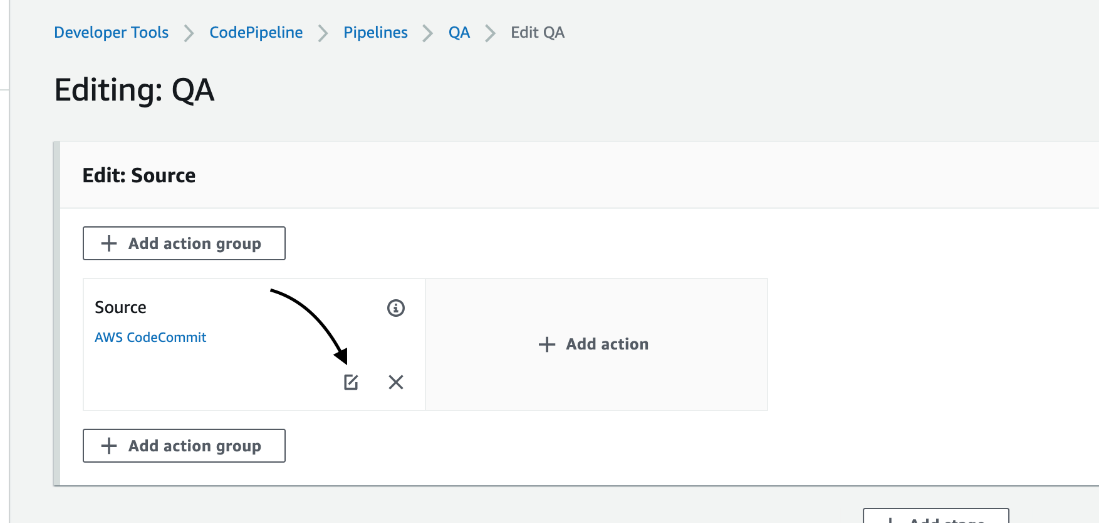
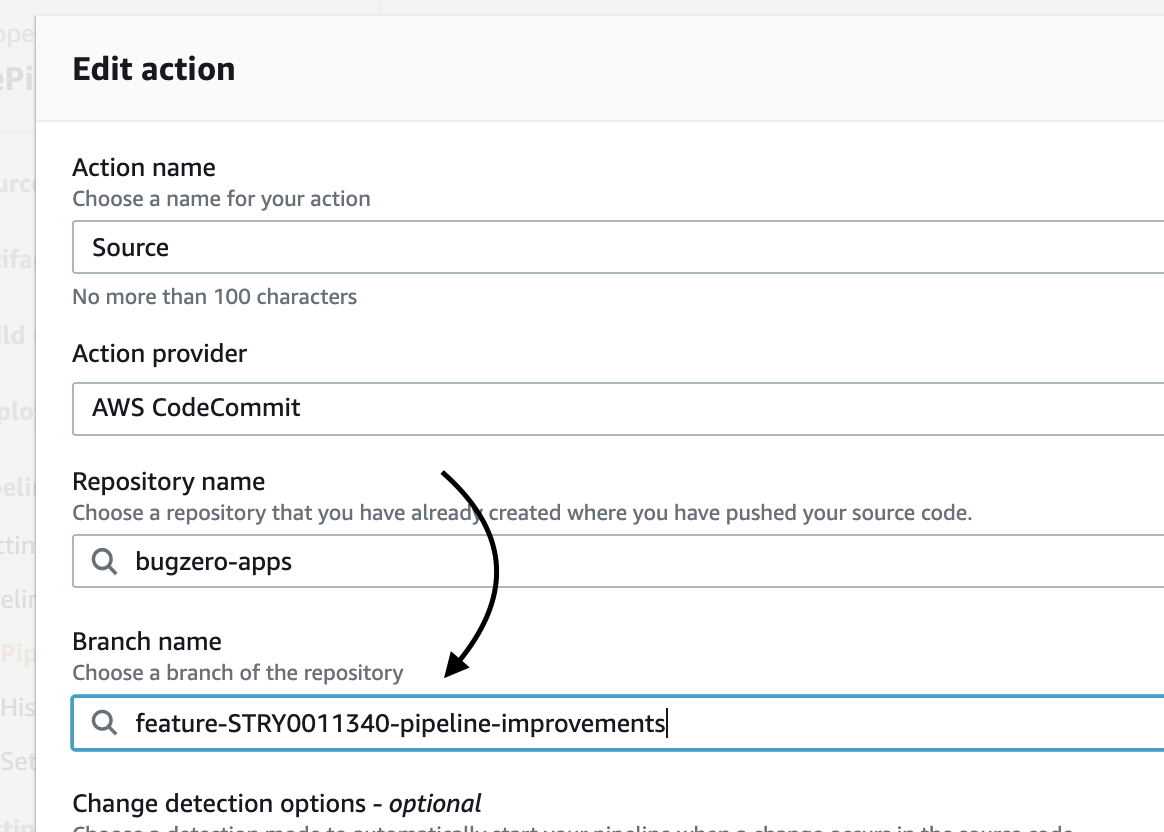
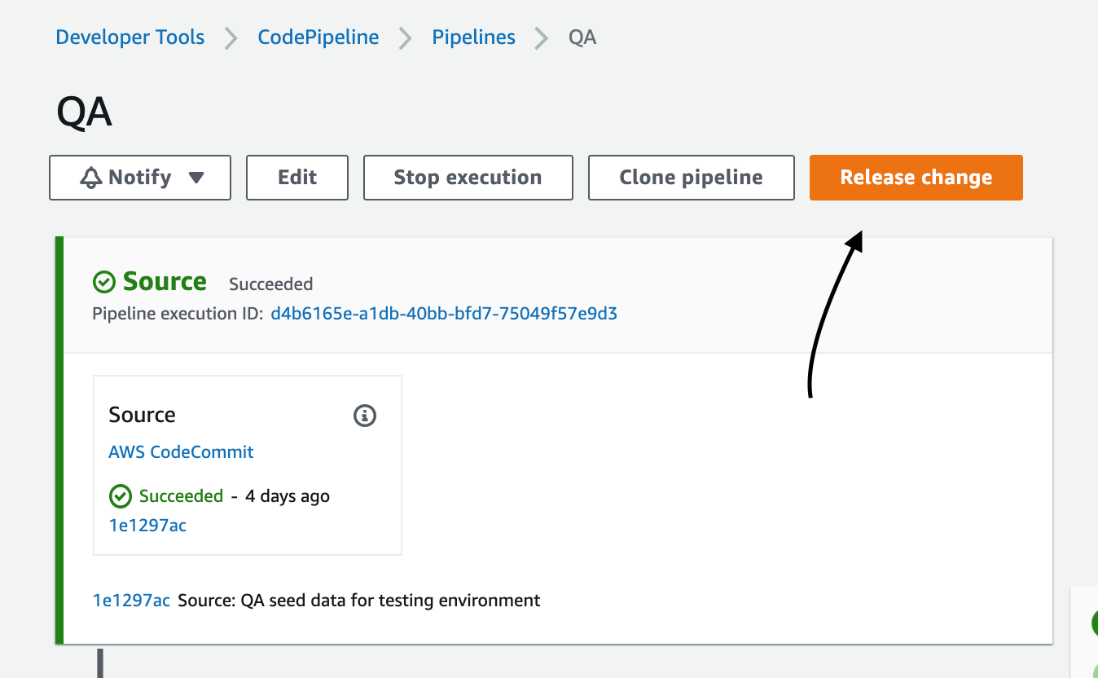
**2. Starting work on a feature/bug:**

When ready to start work on a feature/bug, follow the process:

1. Find the feature/bug story in ServiceNow. If status is not set to "Ready", verify the comments as to what is left to be defined before starting the ticket.
2. Follow **Change Management - Git Workflow for a new feature** section for branch setup, code reviews, and merging.
3. Set story status to **"In progress"**.
4. When complete, set story status to **"Ready for test"**.
5. Add a link to the pull request on CodeCommit in the Work Notes.

**3. Testing a feature/bug:**

When a story is in "Ready for test", follow the process:

1. Review Acceptance criteria to verify the ticket is testable.
2. Put the ticket into status **"Testing".**
3. **(Review Test scenario to make sure this step is necessary) Spin up a test environment to test in QA-Staging environment:**
   1. **Inside the story, click on the Pull request URL.**
   2. **Note the branch name for the pull request for the following steps.**
   3. **Login to AWS (Tools account) and navigate to CodePipeline -> QA (**<https://console.aws.amazon.com/codesuite/codepipeline/pipelines/QA/view?region=us-east-1>**)**
   4. **Click Edit.**
   5. **For “Edit: Source” stage, click “Edit stage”.**
   6. **Inside “Edit: Source” stage, Click the Edit icon button inside the Source Action Group box (AWS CodeCommit)** 
   7. **Change the branch to match the Pull request branch.** 
   8. **Click Done on the bottom, then Done again on the top right, and then Save.**
   9. **Click “Release Change” button on the top right to build the environment.**
   10. **Once the CodePipeline finishes, you can complete your testing at** <https://qa-staging.findbugzero.com>**.** 
4. Review pull request in Work notes. After the code review, add an approval if it passes. If not, leave a comment in the pull request and update the story.
5. If the ticket passes QA, add "Testing notes" describing your testing and why it passes. Set story status to "**Complete"**.
6. If the ticket fails QA, add "Testing notes" describing your testing and why it failed. Set story status back to **"In progress"**.

**4. Deploying a feature/bug:**

When a story is in "Ready for Production, follow the process:

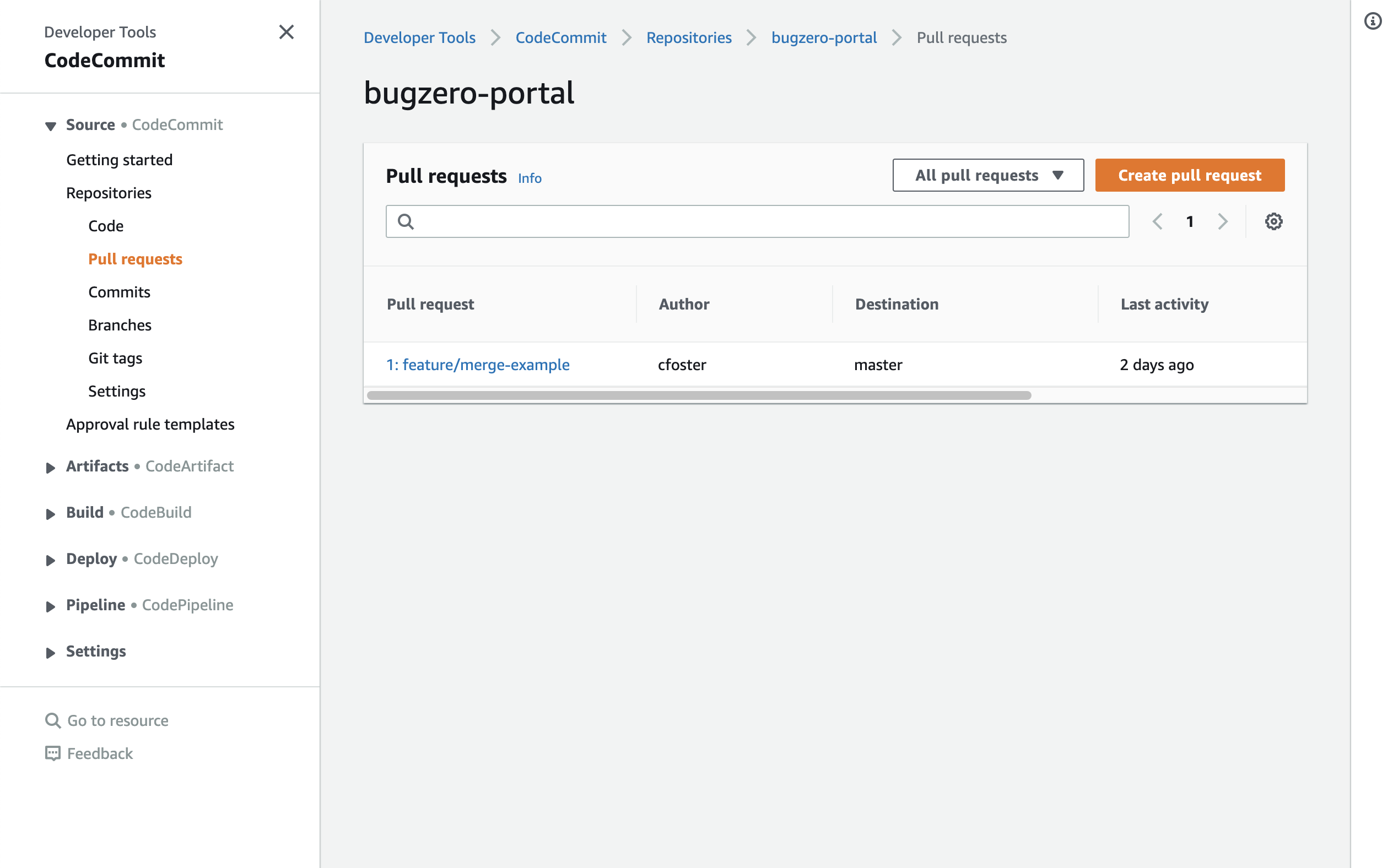
1. If a story is in "Complete" it will be included in the next release cycle.
2. If this story needs a hot fix to be deployed immediately, reach out to the team lead.
3. Upon being successfully verified, the release will be deployed.
4. All tickets in complete will have their "Deployed" attribute set to true automatically via the business rule when a change request is complete.
5. If the feature is not verified, set the status to "In progress" and add "Testing notes" specifying what failed.

### Change Process

All changes are managed in Git via AWS CodeCommit.

#### Git Workflow for a new feature

1. A feature branch (Branch should include all stories needed to test a feature. If more than one story is needed, create an epic, and name the feature off the Epic) is forked off of master branch. (Eg. feature-STRY000323-example or for an epic branch feature-EPIC0001001-aurora-upgrade)
2. Naming of branches will consist of feature-[story|epic #]-[short-descriptor]
3. Code is committed to the feature branch.
4. Code is pushed up daily to the remote repository on AWS CodeCommit.
5. When the feature is complete, a pull request is created using the Approval rule template which requires two peer reviews before a feature may be merged.
6. The feature will not be merged until deployment change request is in progress and a second approval from the change request meeting.
7. Upon successful approval from the change request meeting, the pull request will be merged by the original developer, squashing the commits into one detailed commit for the feature.
8. Delete the feature branch.



(Step 1: AWS CodeCommit interface to create a pull request)

#### Create Pull Request

(Step 2: AWS CodeCommit interface to create a pull request)

#### Create Pull Request

(Step 3: AWS CodeCommit interface to create a pull request)

#### Create Pull Request

(Step 4: AWS CodeCommit interface to create a pull request)

#### Create Pull Request

(Step 6: AWS CodeCommit interface to create a pull request)

#### Create Pull Request

(Step 7: AWS CodeCommit interface to create a pull request)

#### Pre-Commit Checks for Secure coding and Quality Checks

#### All repositories enforce pre-commit checks. This includes:

* Consistent code styling.
* Proper formatting.
* Unused code and other linting errors
* Syntax errors
* Unit Tests
* Services Deployment Process
* Deployment is managed with AWS IAM deployment policies and roles via Serverless/AWS CloudFormation.

#### Services Deployment with AWS Providers

* Decouple credential management from development/deployment
* Automatically use the correct accounts when deploying to different stages
* No need to store credentials locally
* No need to share credentials out of band (e.g. sending slack messages)
* AWS Access Roles, provide an additional layer of security as credentials are generated per deployment and have a short TTL.

#### Services Deployment process for development

1. Restricted via the AWS IAM credentials provisioned for development.
2. Using Serverless framework, deploy the feature branch ready to test.
3. Review changes.

#### Services Deployment process for production

1. Restricted via the AWS IAM credentials provisioned for production.
2. Only code from the master branch that has been peer-reviewed will be deployed.
3. Using Serverless framework, deploy the master branch.
4. Review changes.

#### Separation of Production and Development

* Production credentials are separated via AWS Organizations into its own isolated AWS Account.
* Production access is restricted to authorized personnel only.
* All resources deployed will be isolated into their respective AWS organization.

*Disclaimer: This document is the intellectual property of BugZero. It is confidential and for internal use only. The content is accurate as of the time of publication. Given the evolving nature of technology, the reader is encouraged to use the document to supplement his or her own continuing education on the subject matter.*