PHASE 3

CODEBUILD SETUP

STEP 1: Create S3 Bucket

- Create a new bucket by opening a new console tab → Just give bucket name and leave other settings as default
- (If you forget then you have to redo the Entire codebuild process again after refreshing it 🙂)

STEP 2: Creating a CodeBuild Project

Definition:

> What is AWS CodeBuild?
>
> **AWS CodeBuild** is a fully managed build service that automates the process of compiling source code, running tests, and producing software packages. It eliminates the need to provision and manage dedicated build servers, and you only pay for the actual compute time used during builds.
>
> As part of a continuous integration (CI) workflow, CodeBuild helps detect issues early by automatically testing code changes as soon as they are pushed. This ensures that new code integrates smoothly with the existing codebase, improving software quality and accelerating development.

- Go to CodeBuild in AWS console.

- In build projects menu → Create Build projects

  What is a CodeBuild project?

  A CodeBuild project is basically the blueprint for your CI process. It's where you tell AWS everything it needs to know about how to build your application. This includes things like **where** your code lives (like GitHub), what kind of **environment** you need (Linux or Windows? Java or Python?), exactly what **commands** to run during the build, and where to **store** the results when it's done. Think of it as a recipe that CodeBuild follows every time it needs to build your application.

- Give a project name → Select project type as Default project

> What are the CodeBuild project types?
>
> CodeBuild gives you two main types of projects, each designed for different CI/CD needs:
>
> - **Default project**: This is your standard option that most teams use. It's perfect when you want to manage your entire build process within AWS. You get full control over how your build runs, what goes in, and what comes out - all without leaving the AWS ecosystem.
> - **Runner project**: This option is for teams who already have CI systems like GitHub Actions or GitLab CI but want to tap into the power of CodeBuild's build environment. It's like having CodeBuild do the heavy lifting while your existing CI system orchestrates the overall process.

- Under source → Select source provider as GitHub .
- It will show that you have not connected to GitHub. Click on Manage account credentials. Click on it.

- Select GitHub App for credential type

> What are the Credential types for GitHub?
>
> When connecting to GitHub, CodeConnections gives you a few different options, each with their own trade-offs:
>
> - **GitHub App**: This is generally the simplest and most secure option. AWS manages the application and connection, reducing the need for you to handle tokens or keys directly. It's recommended for most use cases due to its ease of use and enhanced security.
> - **Personal access token**: This method uses a personal access token generated from your GitHub account. You might remember using this for authenticating to GitHub from the terminal. While straightforward, it requires you to manage and rotate tokens, which can be less secure and more operationally intensive.
> - **OAuth app**: This involves setting up an OAuth application in GitHub and configuring CodeConnections to use it. It provides a more granular control over permissions but is more complex to set up compared to GitHub App.

- Click on create a new GitHub connection → Enter a connection name → Click Next → Click Authorize AWS Connector for GitHub → Authenticate → Select username → Connect

  NOTE: You might be wondering why there are *so* many steps just to connect to GitHub. The multi-step process ensures that AWS can securely access your repositories without needing your GitHub password *or* storing sensitive

credentials. This method is much more secure than manually managing tokens that need to be rotated regularly.

- Select newly created connection and click on Save. Now it will show that account is connected to AWS managed GitHub app

  Why save default source credential?

  By saving the GitHub App connection as the default credential, you make it easier to reuse this connection for future CodeBuild projects. This avoids the need to repeat the connection setup process each time you create a new project that uses the same GitHub account.

  When we were taken to different pages to connect to GitHub, that was CodeBuild passing us to another Code service (called **AWS CodeConnections**) behind the scenes.

  > What is AWS CodeConnections?
  >
  > AWS CodeConnections is like a secure bridge between AWS and your external code repositories. Instead of dealing with the headache of managing API keys, tokens (like GitHub's Personal Access Tokens!), or SSH credentials, CodeConnections handles all that authentication complexity for you - so you can focus on building your application.
  >
  > If you'd like, you can open the left hand navigation menu, expand **Settings** at the bottom of the list, and open the **Connections** page. You can manage all connections you set up with CodeConnections there.

- Now we can select the github repositories under Repository dropdown.

STEP 3: Define environment for build to run

(This includes the operating system, runtime, and compute resources)

- Scroll down to Primary source webhook events section
- Untick the checkbox saying “Rebuild every time a code change is pushed to this repository.”
- Under Environment section → Under Provisioning model choose On-demand

> Provisioning model determines how AWS will set up and manage everything needed for your build. Choosing On-demand means AWS will create the resources you need for your build only when you start it, and tear them down when the build is done. This is cost-effective and efficient!
>
> - **Reserved capacity** gives you **dedicated** build resources always at your disposal. It costs more overall but gives you consistent performance and no wait times. Great for teams that are building constantly throughout the day. If **on-demand** is like ordering a taxi, **reserved capacity** is like renting your own car that you can access anytime you need it.

- For environmental image → Select managed image

> Why did we pick managed image?
>
> **Environment image** is like a template for your build environment (just like how AMI's are templates for your EC2 instances). In more technical terms, environment images are pre-configured versions of the build environment so you won't need to install all the software/tools/settings required

> to build a project. We choose **Managed image** here, which means we're using a template that AWS has already created for us. The next few settings we pick underneath this will then tell CodeBuild what kind of image we're looking for.
>
> **Custom image** lets you bring your own Docker image with exactly the tools and configurations your project needs. It's like designing your own custom workspace from scratch - more work to set up, but perfectly tailored to your specific requirements.

- For Compute type → Select EC2

> Why did we pick EC2?
>
> **Compute** sets up the servers that will actually run the commands and do the work for your project's build! Your project's build will run on **Amazon EC2** instances, which are more flexible and powerful than **AWS Lambda** functions.

- For Running mode → Container
- For OS → Select Amazon Linux , Runtime → Standard,
- Image → aws/codebuild/amazonlinux-x86_64-standard:5.0
- Image Version -Always use the latest image for this runtime version
- Service Role- New service role
- Under Buildspec section → Select Use a buildspec file → Leave the name field as default without changing it. (Ensure project has this .yml file in GitHub. Follow Step 4 for its creation)

> What is buildspec.yml?

The buildspec.yml file is like a detailed instruction manual for CodeBuild. Placed in the root of your repository, it tells CodeBuild exactly what to do at each stage of the build process - what tools to install, what commands to run, and what files to package up when it's done.

CodeBuild automatically looks for a file named buildspec.yml in the root directory of your source code. If it finds one, it uses it to execute the build. If not, the build will fail.

> CodeBuild reads your buildspec.yml file like a step-by-step instruction manual. It goes through each phase in order - install, pre_build, build, then post_build - running the commands you've specified in each section.
>
> The beauty of using buildspec.yml is that your build process is defined as code right alongside your application. This means your build process can be versioned, reviewed, and evolved just like any other part of your codebase. Before the era of continuous integration (CI), you would have to manually run a bunch of commands to build your

> > application - buildspec.yml automates this process!

- Batch Configuration section remain untouched

> What is Batch configuration?
>
> Batch configuration lets you run multiple builds at once as part of a single batch job. It's like being able to cook several dishes simultaneously instead of one after another.
>
> This becomes super useful when you need to test your code across different environments (like various operating systems or browser configurations) or when you want to parallelize parts of your build process to save time. While we won't use it in this project, it's a powerful feature for more complex workflows.

- Scroll down to Artifacts section to configure where Codebuild will store the build artifacts

> What are build artifacts?
>
> **Build artifacts** are the tangible outputs of your build process. They're what you'll actually deploy to your servers or distribute to users. That's why storing them properly in S3 is so important - they're the whole reason we're running the build in the first place.

> For the project, the build process should create one build artifact that packages up everything a server could need to host our web app in one neat bundle. This bundle is a ZIP file.
>
> Note: The build process will create a .zip file (a packaged Python Flask web application) as the build artifact, but artifacts could be executables, libraries, documentation, or any output the build creates.

- For Type → Select Amazon S3

  Why store artifacts in Amazon S3?

  The compiled applications, libraries, or any output files from the build need a safe, accessible home after the build finishes. S3 is perfect for this - it's a highly reliable and scalable storage solution that's also in the AWS environment (which makes the artifact easily accessible for deployment later).

- Choose bucket name from dropdown → Give a name for the artifact → Under Artifacts packaging select Zip

> Why package artifacts as a Zip file?
>
> Packaging artifacts as a Zip file is a small detail that's actually quite useful!
>
> - **Smaller size**: Zip compression can significantly reduce the size of your artifacts, which means faster uploads to S3, less storage costs, and quicker downloads when you need to use them.
> - **Organization**: Instead of managing multiple individual files, you get one tidy package. It's like putting all your

> vacation photos in a single album rather than having them scattered across your phone.
>
> - **Simplicity**: When it comes time to deploy your application or share it with teammates, having a single zip file makes the process much more straightforward - just download one file and you have everything you need.

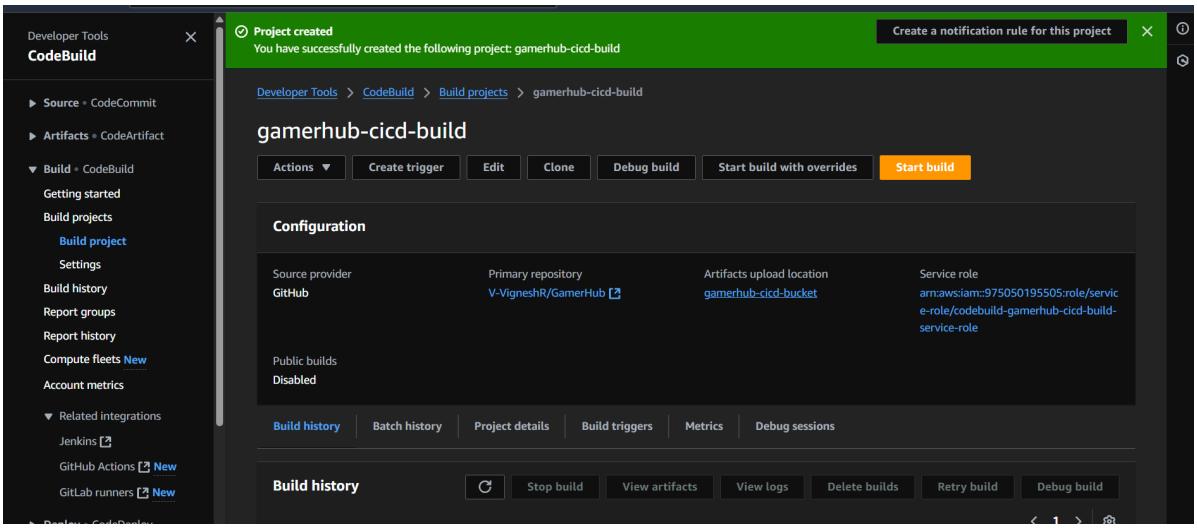
- Scroll down to Logs and ensure CloudWatch logs box is checked

  What are CloudWatch logs?

  Amazon CloudWatch Logs is a monitoring service that collects and tracks logs from AWS services. In this project, CloudWatch will record everything that happens during the build process, including the commands that are run, the output of those commands, and any errors that occur. This is incredibly useful for debugging and understanding what went wrong if a build fails.
- Leave group name as default for this project.
- Give group name like : aws/codebuild/gamerhub-cicd-logs

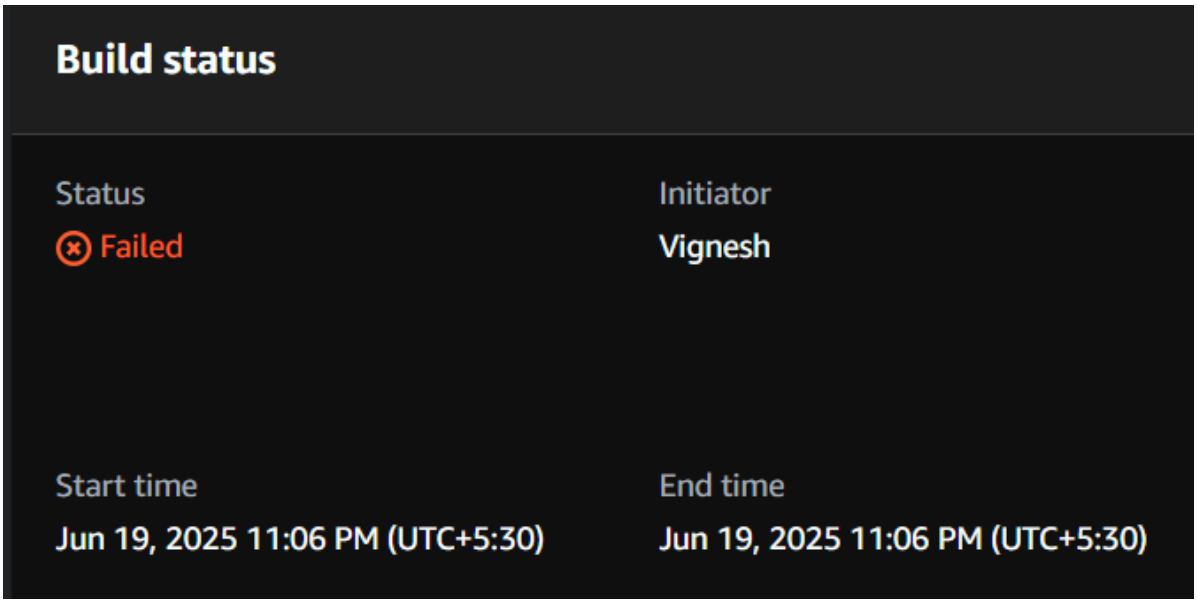
> Why set a custom Group name?
>
> Setting a custom Group name for CloudWatch logs might seem like a small detail, but it's helpful to keep things organized. By using a specific name like /aws/codebuild/gamerhub-cicd-logs, you're essentially creating a dedicated folder for all logs related to this project.
>
> This becomes super helpful as your AWS environment grows. Imagine having hundreds of projects and services to track! With custom group names, you can instantly filter and find the logs you need with this classic naming convention.

- Click on Create project (Finally)

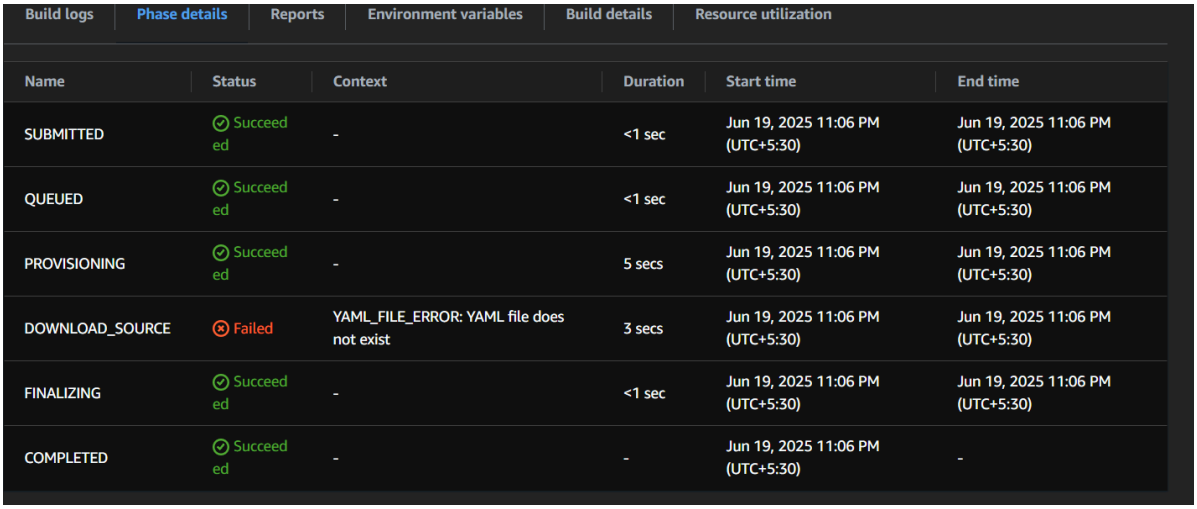


STEP 4 : Run the Build and Troubleshoot Failures

- Go to newly created CodeBuild project
- Click the Start Build button
- Initially the build fails



- The Phase details shows:



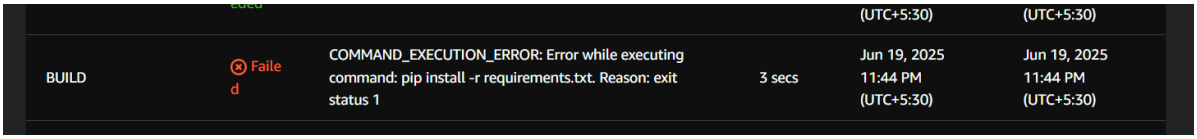
> Why this error?
>
> This is expected. This error simply means CodeBuild couldn't find the buildspec.yml file in your GitHub repository, which makes sense because we haven't created it yet.
>
> This is an important learning moment - without this file, CodeBuild doesn't know how to build your project.

- Go to VSCode → Create a buildspec.yml in root folder
- Enter this:

```
version: 0.2

phases:
  install:
    runtime-versions:
      python: 3.12  # Replace with your Flask app's Python version

  pre_build:
    commands:
      - echo Initializing environment
      - export CODEARTIFACT_AUTH_TOKEN=`aws codeartifact get-authorization-
      - pip config set global.index-url https://aws:$CODEARTIFACT_AUTH_TOKEN@

  build:
    commands:
      - echo Build started on `date`
      - pip install -r requirements.txt

  post_build:
    commands:
      - echo Build completed on `date`
      - echo Packaging Flask app...
      - zip -r flask-app.zip .  # Package all files as an artifact

artifacts:
  files:
    - flask-app.zip
  discard-paths: no
```

STEP 5 : Verify Successful Build and Artifacts

- Re-run the build process in CodeBuild. Build fails:



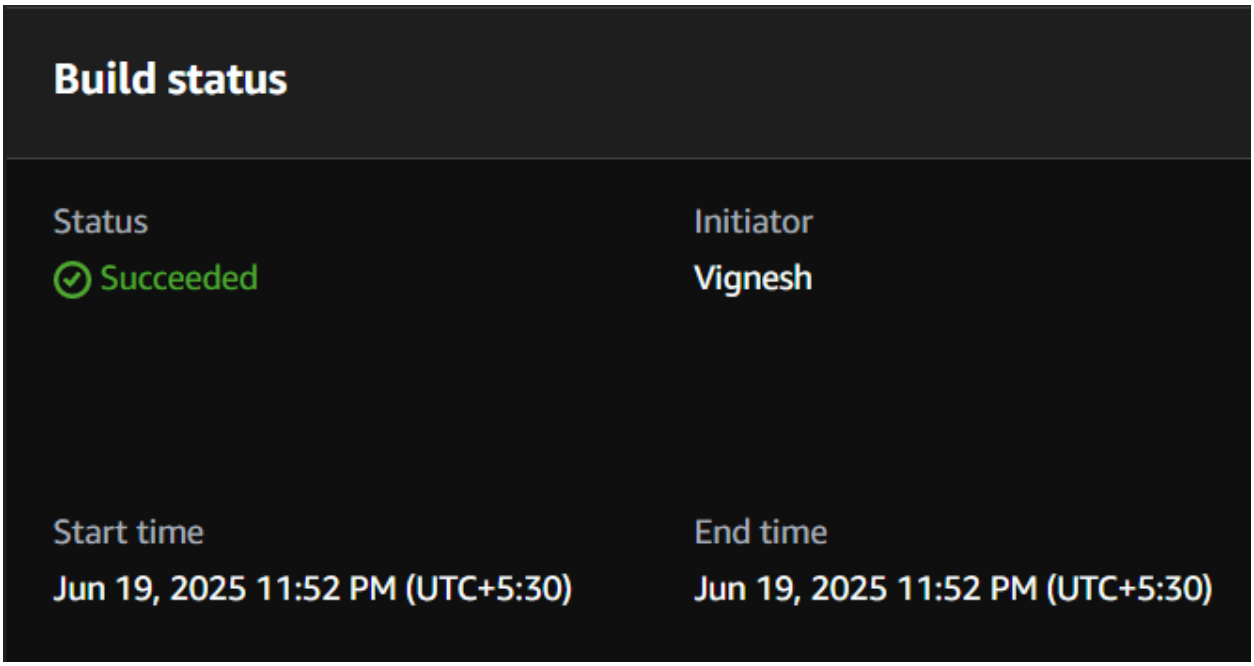
  This usually happens because our CodeBuild service role **doesn't have permission to access CodeArtifact,** which is needed to download project dependencies.

  To fix this, we need to grant CodeBuild's IAM role the permission to access CodeArtifact.

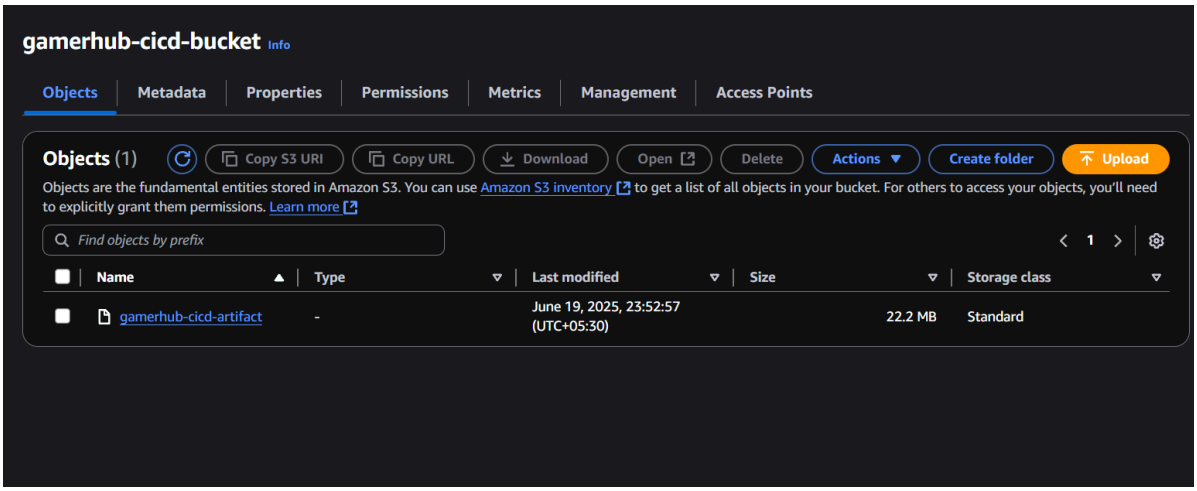
- Go to IAM console → Roles → search for Codebuild role that was created before. Open it and click on Add permissions → Attach policies → search for the codeartifact-consumer-policy created earlier in the project → click on Add permissions.
- Retry the build
- Wow it was succeeded !

Build logs | Phase details | Reports | Environment variables | Build details | Resource utilization

Name	Status	Context	Duration	Start time	End time
SUBMITTED	Succeeded	-	<1 sec	Jun 19, 2025 11:52 PM (UTC+5:30)	Jun 19, 2025 11:52 PM (UTC+5:30)
QUEUED	Succeeded	-	<1 sec	Jun 19, 2025 11:52 PM (UTC+5:30)	Jun 19, 2025 11:52 PM (UTC+5:30)
PROVISIONING	Succeeded	-	5 secs	Jun 19, 2025 11:52 PM (UTC+5:30)	Jun 19, 2025 11:52 PM (UTC+5:30)
DOWNLOAD_SOURCE	Succeeded	-	3 secs	Jun 19, 2025 11:52 PM (UTC+5:30)	Jun 19, 2025 11:52 PM (UTC+5:30)
INSTALL	Succeeded	-	<1 sec	Jun 19, 2025 11:52 PM (UTC+5:30)	Jun 19, 2025 11:52 PM (UTC+5:30)
PRE_BUILD	Succeeded	-	15 secs	Jun 19, 2025 11:52 PM (UTC+5:30)	Jun 19, 2025 11:52 PM (UTC+5:30)
BUILD	Succeeded	-	17 secs	Jun 19, 2025 11:52 PM (UTC+5:30)	Jun 19, 2025 11:52 PM (UTC+5:30)
POST_BUILD	Succeeded	-	2 secs	Jun 19, 2025 11:52 PM (UTC+5:30)	Jun 19, 2025 11:52 PM (UTC+5:30)
UPLOAD_ARTIFACTS	Succeeded	-	1 sec	Jun 19, 2025 11:52 PM (UTC+5:30)	Jun 19, 2025 11:52 PM (UTC+5:30)
FINALIZING	Succeeded	-	<1 sec	Jun 19, 2025 11:52 PM (UTC+5:30)	Jun 19, 2025 11:52 PM (UTC+5:30)
COMPLETED	Succeeded	-	-	Jun 19, 2025 11:52 PM (UTC+5:30)	-

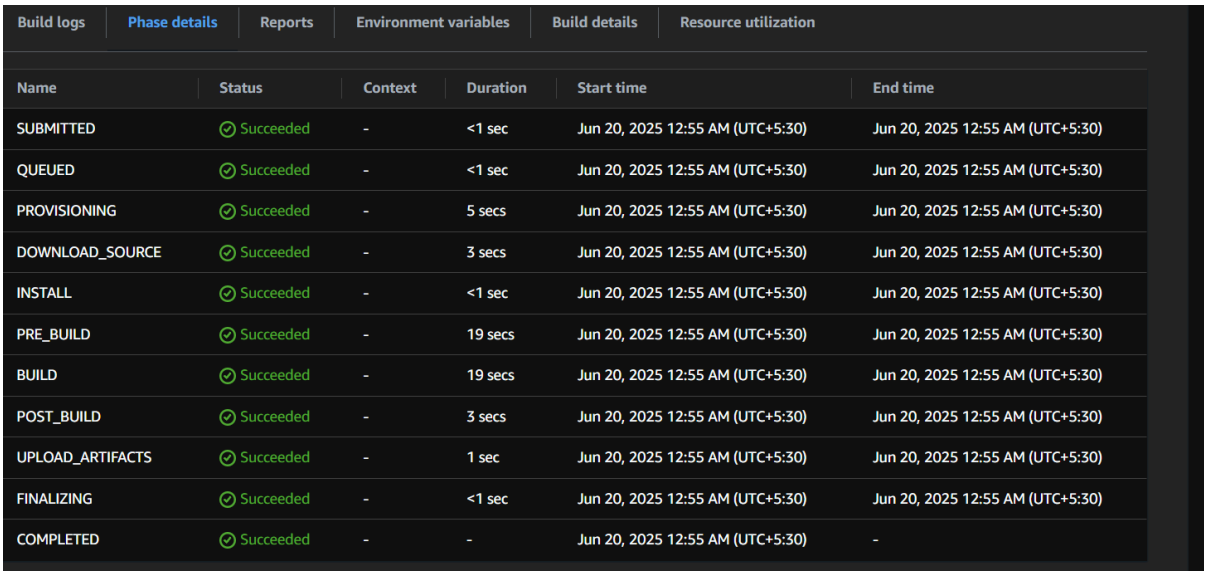


- Verify S3 bucket if the zip file is available:



> Why check for build artifacts in S3?
>
> By confirming that an artifact exists in your S3 bucket, you know that:
>
> - Your code was successfully compiled and packaged
> - The build process completed without errors
> - The artifact was properly uploaded to its destination
> - The artifact is now available for the next step in your process (like deployment)
> - Check by downloading the zip file and you can see that the web app's directory contents are available.

NOTE: The next time we re-build, the build time is reduced: (CodeArtifact cache being used)



Build logs	Phase details	Reports	Environment variables	Build details	Resource utilization

Name	Status	Context	Duration	Start time	End time
SUBMITTED	Succeeded	-	<1 sec	Jun 20, 2025 12:55 AM (UTC+5:30)	Jun 20, 2025 12:55 AM (UTC+5:30)
QUEUED	Succeeded	-	<1 sec	Jun 20, 2025 12:55 AM (UTC+5:30)	Jun 20, 2025 12:55 AM (UTC+5:30)
PROVISIONING	Succeeded	-	5 secs	Jun 20, 2025 12:55 AM (UTC+5:30)	Jun 20, 2025 12:55 AM (UTC+5:30)
DOWNLOAD_SOURCE	Succeeded	-	3 secs	Jun 20, 2025 12:55 AM (UTC+5:30)	Jun 20, 2025 12:55 AM (UTC+5:30)
INSTALL	Succeeded	-	<1 sec	Jun 20, 2025 12:55 AM (UTC+5:30)	Jun 20, 2025 12:55 AM (UTC+5:30)
PRE_BUILD	Succeeded	-	19 secs	Jun 20, 2025 12:55 AM (UTC+5:30)	Jun 20, 2025 12:55 AM (UTC+5:30)
BUILD	Succeeded	-	19 secs	Jun 20, 2025 12:55 AM (UTC+5:30)	Jun 20, 2025 12:55 AM (UTC+5:30)
POST_BUILD	Succeeded	-	3 secs	Jun 20, 2025 12:55 AM (UTC+5:30)	Jun 20, 2025 12:55 AM (UTC+5:30)
UPLOAD_ARTIFACTS	Succeeded	-	1 sec	Jun 20, 2025 12:55 AM (UTC+5:30)	Jun 20, 2025 12:55 AM (UTC+5:30)
FINALIZING	Succeeded	-	<1 sec	Jun 20, 2025 12:55 AM (UTC+5:30)	Jun 20, 2025 12:55 AM (UTC+5:30)
COMPLETED	Succeeded	-	-	Jun 20, 2025 12:55 AM (UTC+5:30)	-

_____PHASE 3 COMPLETED_____