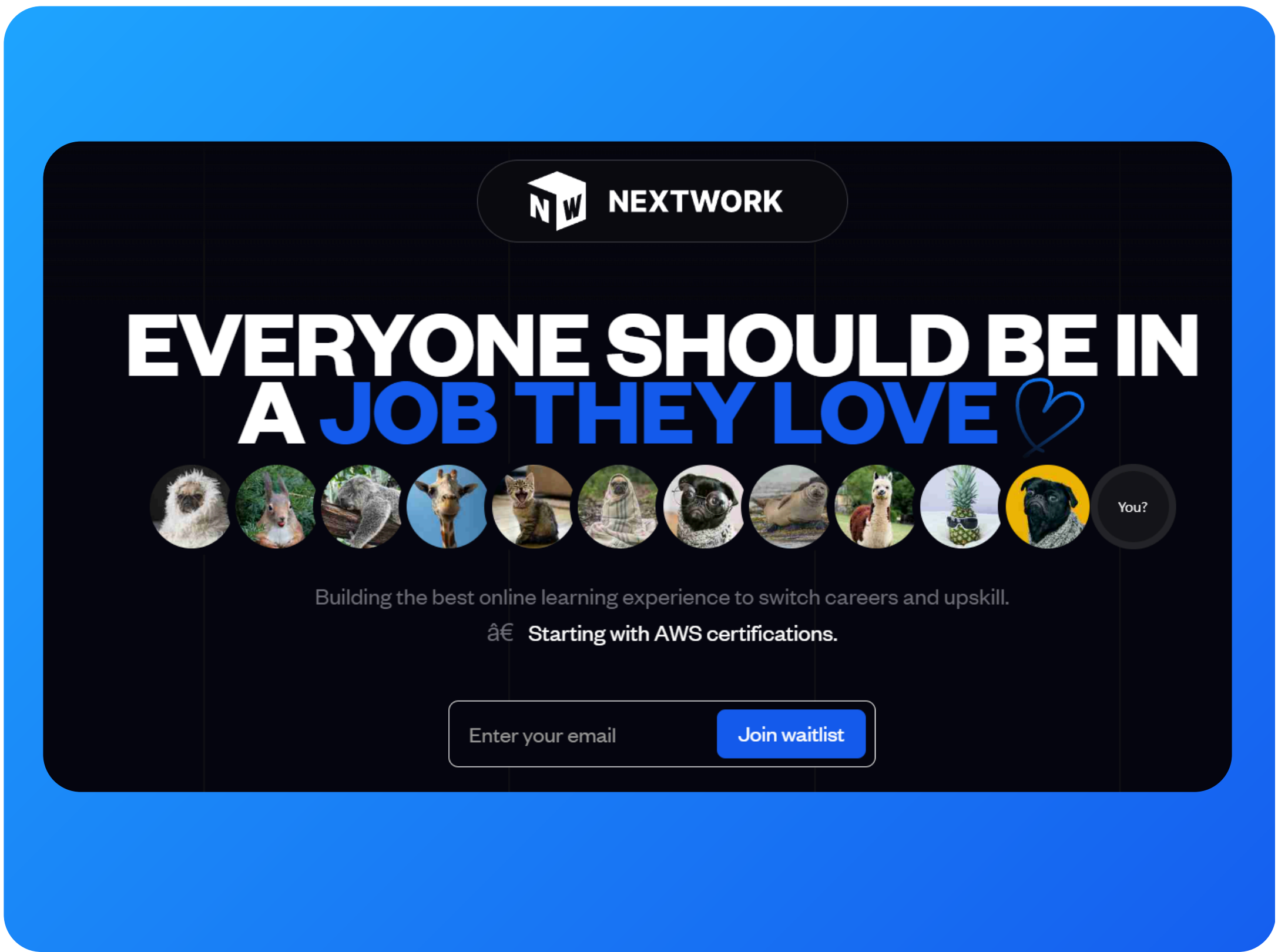
NextWork.org

CLOUD DEVOPS: FROM DEVELOPMENT TO DEPLOYMENT

Kanika Mathur
github.com/KanikaGenesis



Kanika Mathur
github.com/KanikaGenesis

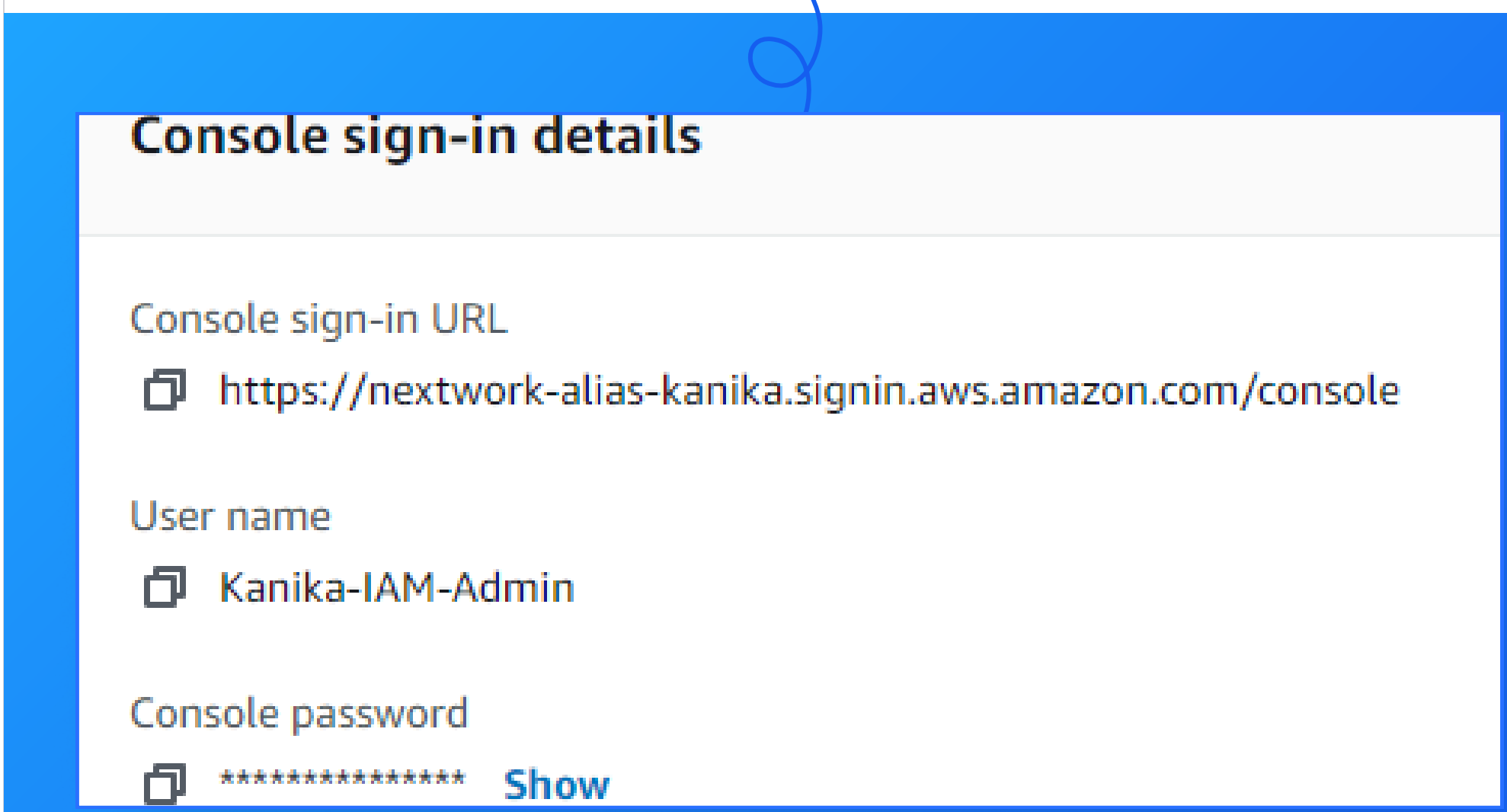
Set up an IAM User

An IAM user is an additional user that gets access to my AWS Account's resources. When creating a User, I can specify in detail the level of access it has to my account's resources and services.

- It's important to create IAM users because the root user is vulnerable to security breaches that could result in my billing information being accessed without my permission.

- I created an IAM user with Administrator Access. This means my IAM user is allowed to perform all possible actions to all the resources in my account.

A new IAM user set up for my AWS Account



Kanika Mathur
github.com/KanikaGenesis

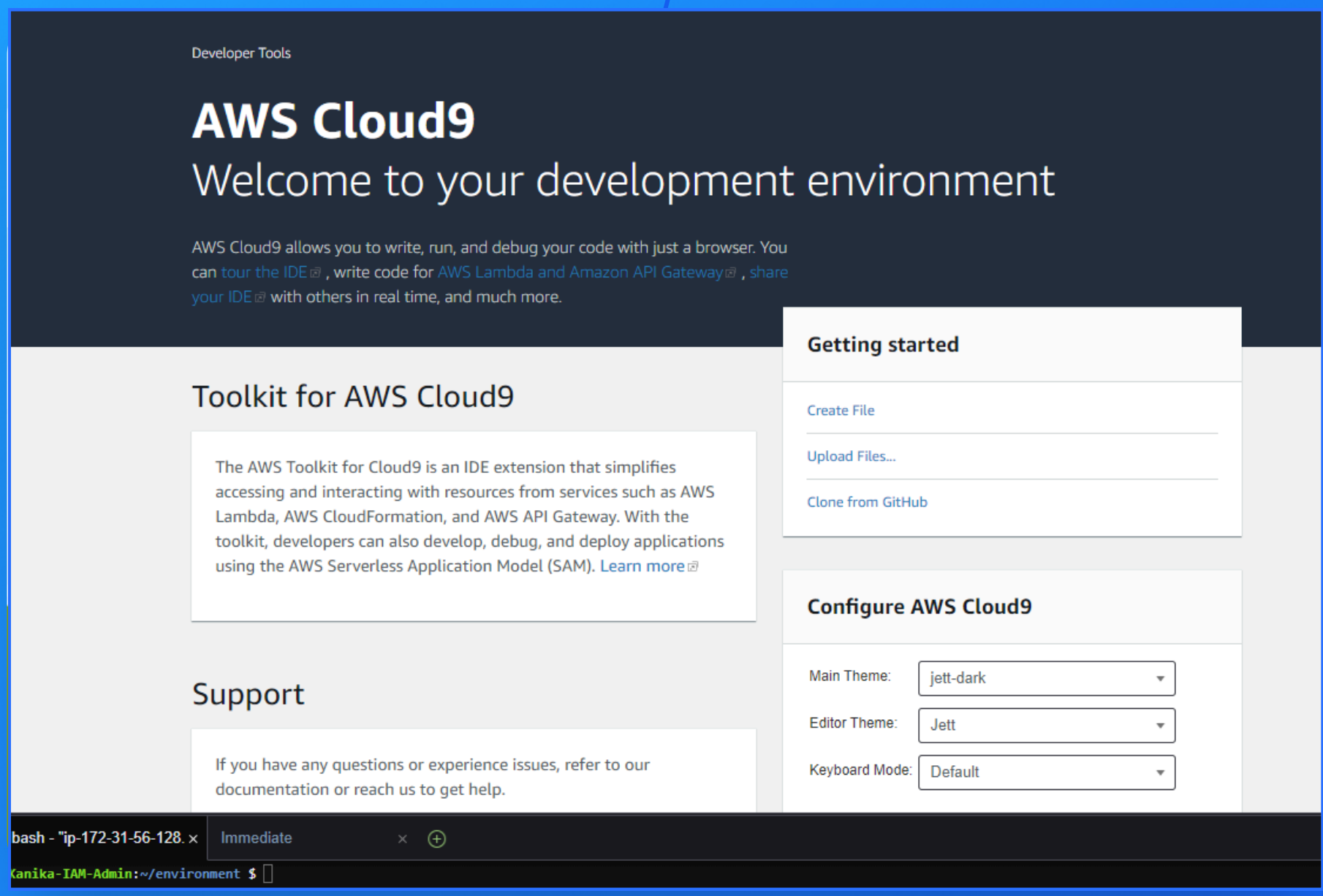
Launch a Cloud9 IDE

- An IDE is an environment for developers to write, manage and debug their code. Just like how Google Docs is a companion of writers to create documents, IDEs are companions for developers to build their application/ edit their code.

- I used AWS Cloud9 to launch an environment. An environment means a set of resources that work together to help me build a piece of software/ application. Every application has its own needs e.g. dependencies, libraries, tools, etc. This means the environment that I'm creating will be tailored to the web application I build.

- Using Cloud9 meant that I did not need to download this software in order to use an IDE.

My Cloud9 IDE!




Kanika Mathur
github.com/KanikaGenesis

Install Maven & Java

- Maven is a tool that helps developers with the building of their software, automating the steps required for my application to become a final product that machines e.g. computers can actually run.
- Maven is required in this project because I am building a web app in a specific programming language which will not be able to build on its own without the help of Maven.

- Java is the specific programming language I am using to create my web application.
- Java is required in this project because it is a versatile tool for creating different applications, including web apps.
- The Java version I'm using for this project is Amazon Coretto 8.

I used terminal commands to install Maven and Java

```
Dependency Installed:
  java-1.8.0-amazon-corretto.x86_64 1:1.8.0_412.b08-1.amzn2

Complete!
Kanika-IAM-Admin:~/environment $ export JAVA_HOME=/usr/lib/jvm/java-1.8.0-amazon-corretto.x86_64
Kanika-IAM-Admin:~/environment $ export PATH=/usr/lib/jvm/java-1.8.0-amazon-corretto.x86_64/jre/bin/:$PATH

Kanika-IAM-Admin:~/environment $ java -version
openjdk version "1.8.0_412"
OpenJDK Runtime Environment Corretto-8.412.08.1 (build 1.8.0_412-b08)
OpenJDK 64-Bit Server VM Corretto-8.412.08.1 (build 25.412-b08, mixed mode)
Kanika-IAM-Admin:~/environment $ mvn -v
Apache Maven 3.5.2 (138edd61fd100ec658bfa2d307c43b76940a5d7d; 2017-10-18T07:58:13Z)
```


Kanika Mathur
github.com/KanikaGenesis

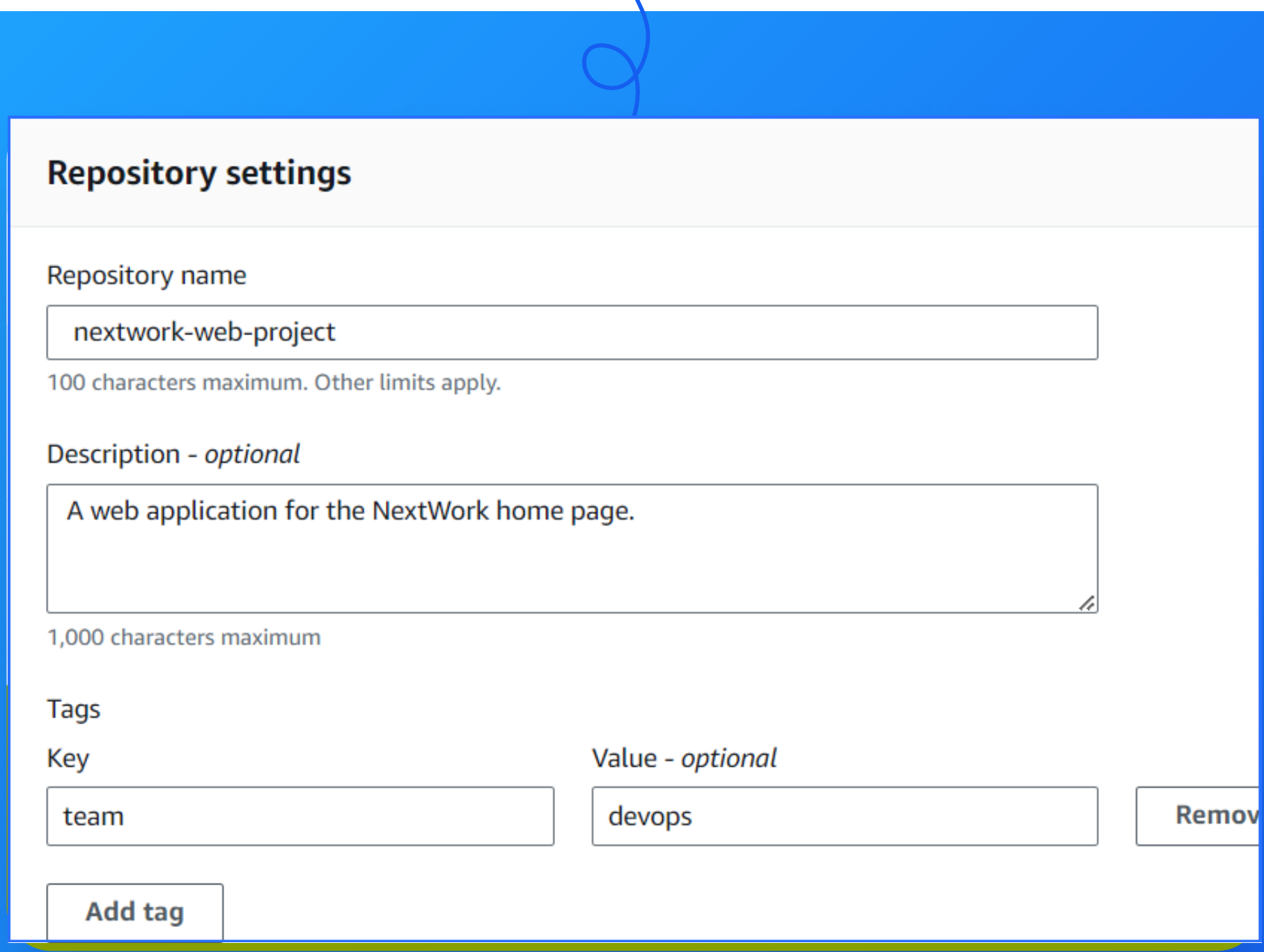
Create a Git repository

- Git is a version control and code management system that helps developers to tracking their changes and collaborating on code together.

- A Git repository is like a folder that contain all of the application/ project's file in one place.

- To create a Git repository in the cloud, I used CodeCommit.

My setup page for a CodeCommit repo



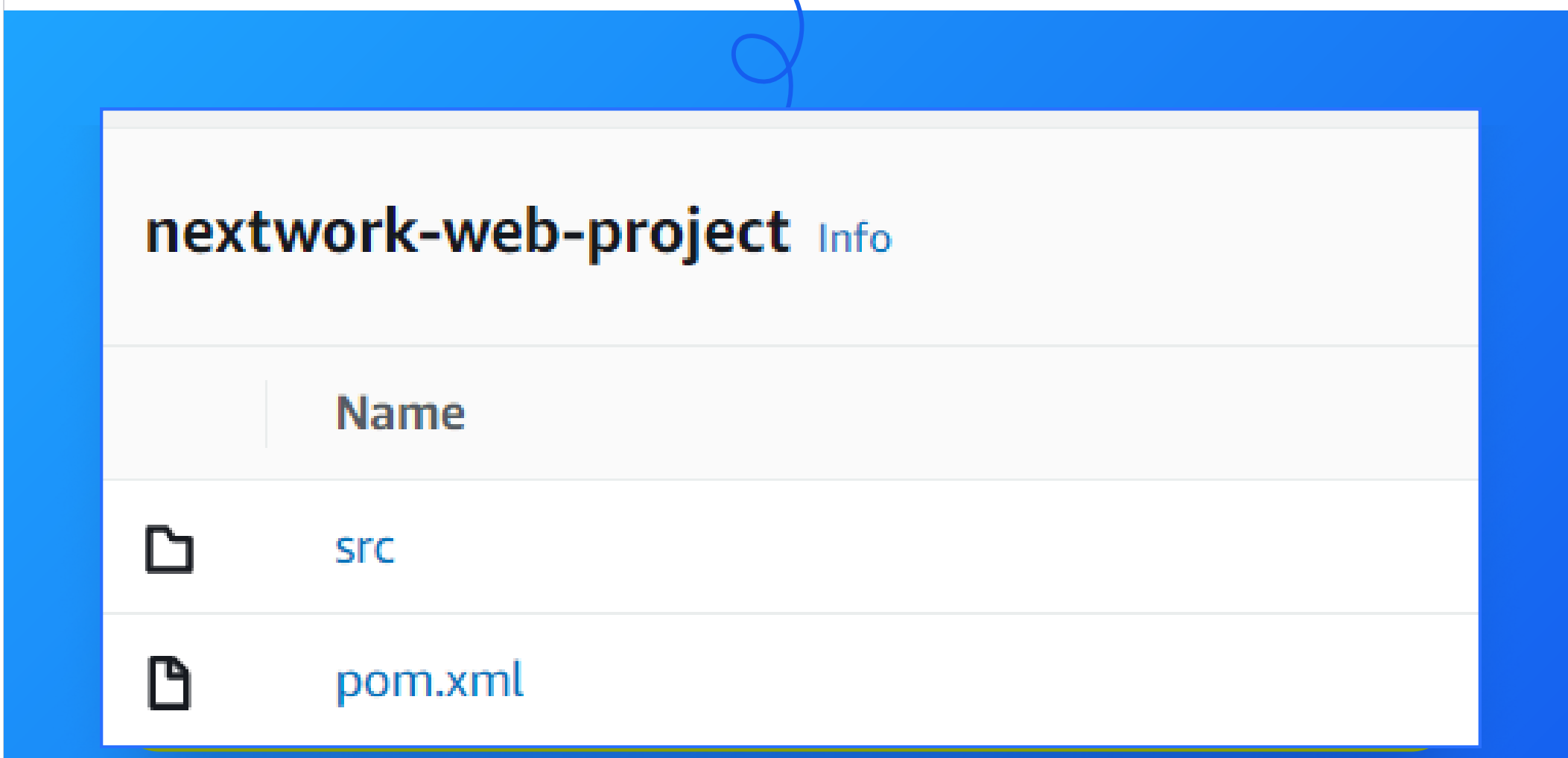

Kanika Mathur
github.com/KanikaGenesis

My first commit

- I initialised a Git repo in my web application by running the command **git init -b main**.

- To commit and push my code, I will have to run three different commands in order:
- **git add** places the file that I've created/edited in a staging area i,e. preparing them to be saved.
- **git commit** is like pushing a "save" button that confirms my changes.
- **git push** sends my changes upstream to my remote origin i,e. CodeCommit repository I've set up.

Files I committed showing up in my CodeCommit repo!



Kanika Mathur
github.com/KanikaGenesis

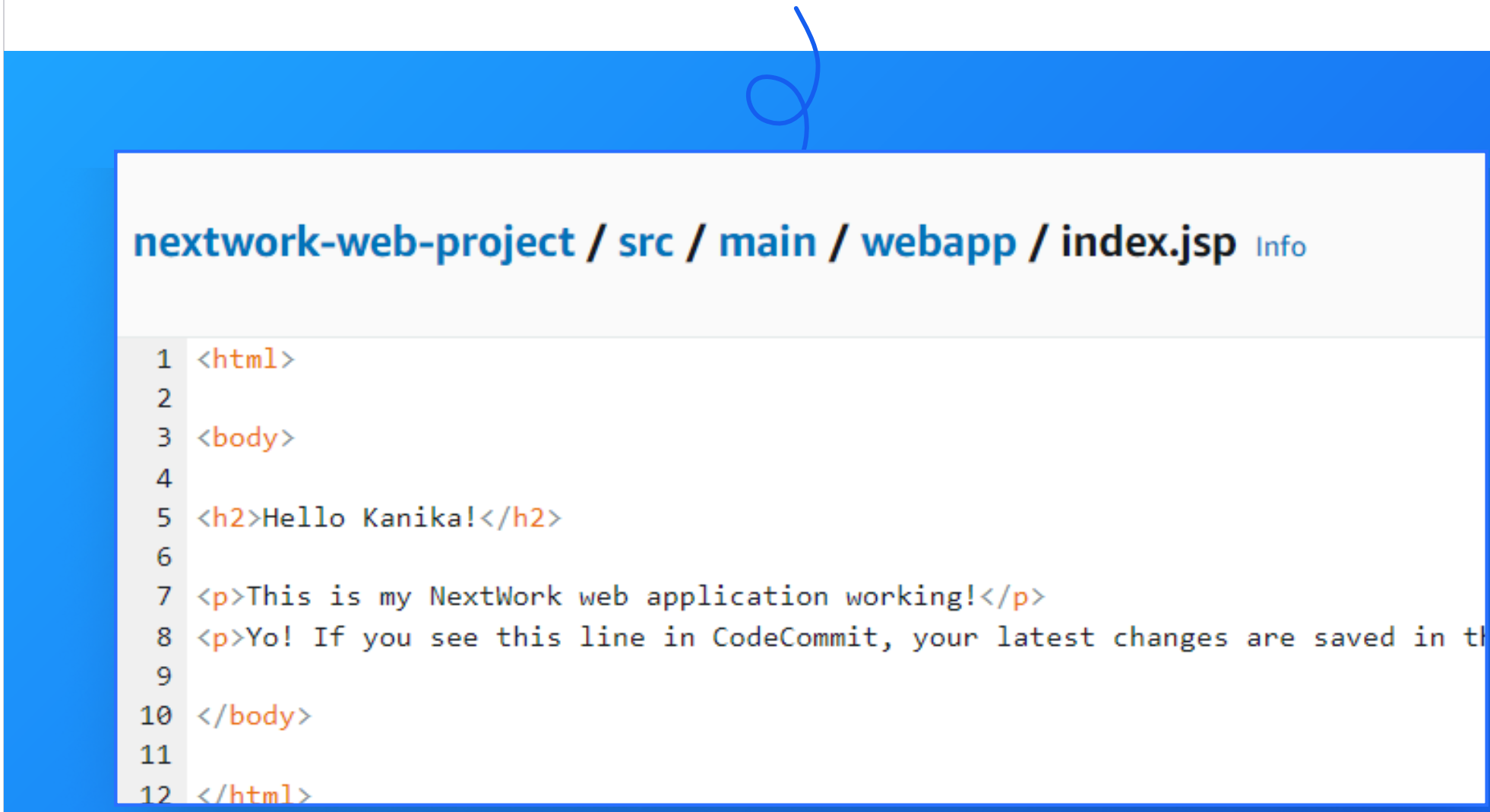
Git in action

- I wanted to see Git working in action, so I updated my index.jsp file by adding two new lines.

- Then I tried seeing these changes in my CodeCommit repository, but this didn't work because I had only saved these changes in my local repository without pushing the changes upstream.

- I finally saw the changes in my CodeCommit repository after running the same three Git commands in my Cloud9 terminal:
  - git add .
  - git commit
  - git push

My updated index.jsp file showing up in CodeCommit!

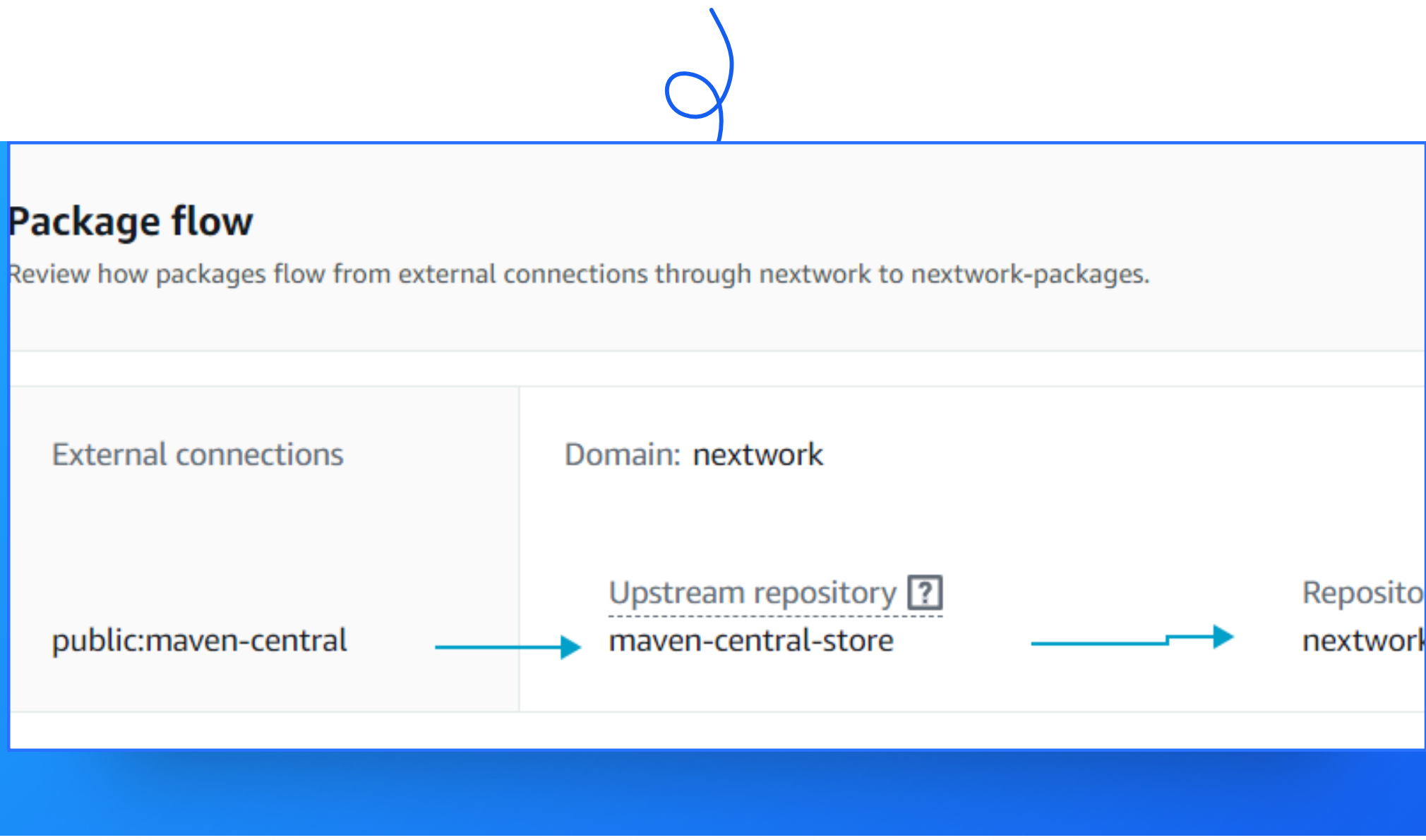


Kanika Mathur
github.com/KanikaGenesis

Create a repository

- CodeArtifact is a service that sits in the CI/CD pipeline. I am using it today to store backup copies of packages (importantly, dependencies) relevant to my java web app.
- The reason why I'm using CodeArtifact for my web app is for security/risk management and continuity. Even if public packages or dependencies of my project are no longer available, there is a copy in my AWS CodeArtifact repositories to make sure I can keep developing my web app.
- Instead of a single repository, there are actually three connected repositories that Maven uses to fetch packages.
- The first is my local repository, which Maven checks for packages/dependencies for my web app.
- The second is my public upstream repository which Maven will check next if the package is not in the local repository.
- The third is the Maven Central Repository, which is a public repository with the greatest collection of packages for Java application. However, Maven will only visit this repository if the first two do not have the packages/ dependencies it is looking for (due to high traffic going into Maven Central Repository).

Package flow illustrating the connections between the three repositories.



Kanika Mathur
github.com/KanikaGenesis

Connecting my project to CodeArtifact

- Next, I connected my web app project (via my Cloud9 IDE) to CodeArtifact so that CodeArtifact knows which project it is going to storing dependencies for!

- I created a new file, settings.xml, in my web app. settings.xml is the file that will give Maven instructions on WHERE to find the dependencies Maven will need to fetch, and HOW Maven will get access to these repositories that are storing these dependencies.

- The code I pasted into settings.xml were provided by CodeArtifact, so I did not have to write from scratch. The snippets of code stores authentication tokens to CodeArtifact and defines when Maven will visit which repository, plus where Maven should visit to find backup local repositories (optional).

My settings.xml file.

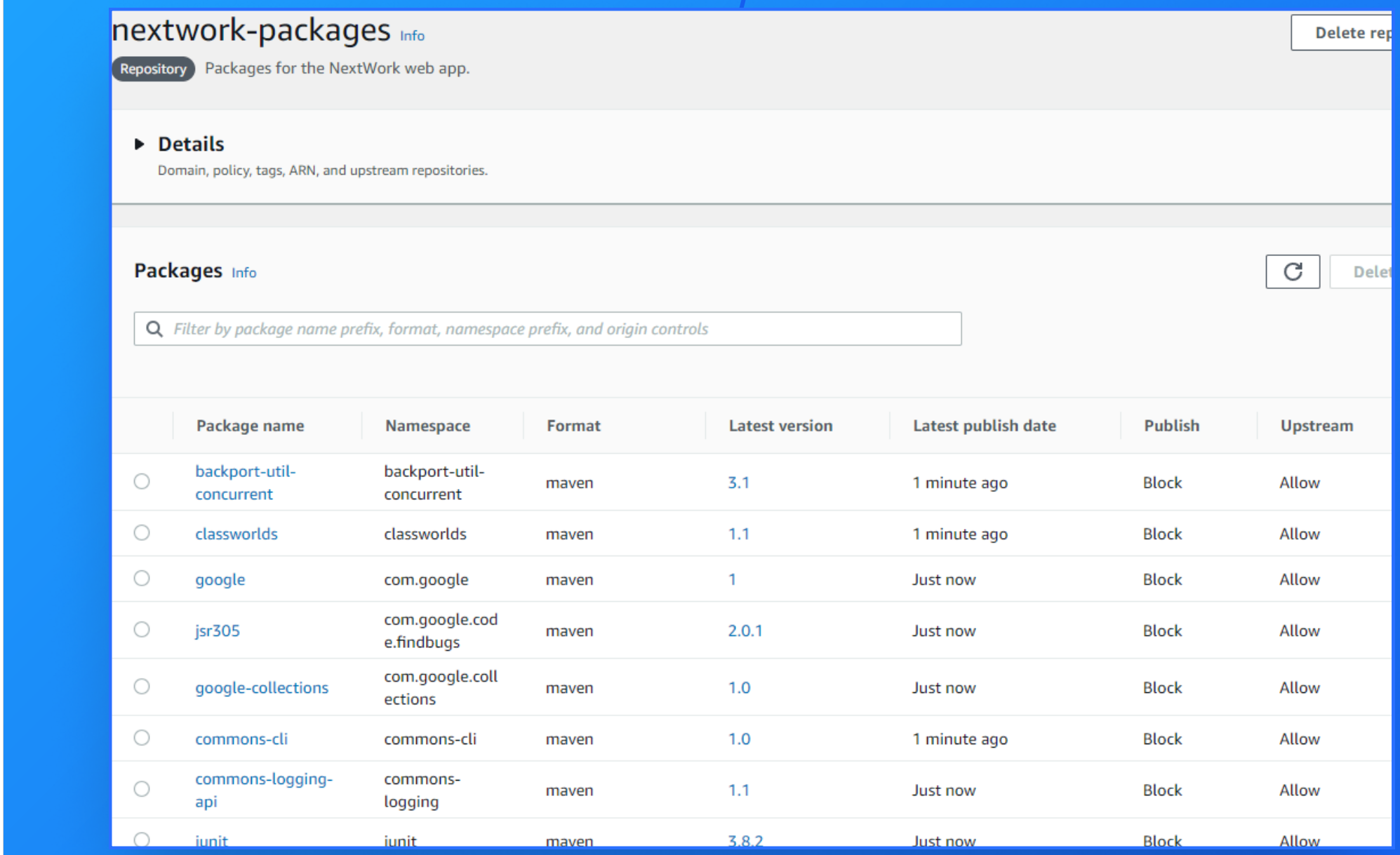
```
<settings>
    <servers>
  <server>
    <id>nextwork-nextwork-packages</id>
    <username>aws</username>
    <password>${env.CODEARTIFACT_AUTH_TOKEN}</password>
  </server>
</servers>

<profiles>
  <profile>
    <id>nextwork-nextwork-packages</id>
    <activation>
      <activeByDefault>true</activeByDefault>
    </activation>
    <repositories>
      <repository>
        <id>nextwork-nextwork-packages</id>
        <url>https://nextwork-991380288324.d.codeartifact.us-east-1.amazonaws.com/maven/nextwork-packages/</url>
      </repository>
    </repositories>
  </profile>
</profiles>

<mirrors>
  <mirror>
    <id>nextwork-nextwork-packages</id>
    <name>nextwork-nextwork-packages</name>
    <url>https://nextwork-991380288324.d.codeartifact.us-east-1.amazonaws.com/maven/nextwork-packages/</url>
```


Kanika Mathur
github.com/KanikaGenesis

Test the connection

- To test the connection between Cloud9 and CodeArtifact, I compiled my web app. Compiling means translating my web application's code into machine code that servers can actually understand and run.
- After compiling, I checked my local repository and saw that my local repository now has pages and pages of packages inside! This means Maven has now grabbed packages from the upstream repository/Maven Central Repository and installed/ kept a copy locally.

My web app's packages popping up in my local repository.




Kanika Mathur
github.com/KanikaGenesis

Create IAM policies

- I also created an IAM policy because other services in my CI/CD pipeline e.g. CodeBuild, CodePipeline will be needing access to the packages/ dependencies stored in CodeArtifact. By default, these services do NOT have access - so they need to be granted access through an IAM Policy.

- I defined my IAM policy using JSON. This policy will enable the policy holder to get authorization token (i.e. access to CodeArtifact), fetch packages stored in CodeArtifact's repositories.

A peek at the policy that will provide access to CodeArtifact

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codeartifact:GetAuthorizationToken",
                "codeartifact:GetRepositoryEndpoint",
                "codeartifact:ReadFromRepository"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "sts:GetServiceBearerToken",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "sts:AWSServiceName": "codeartifact.amazonaws.com"
                }
            }
```

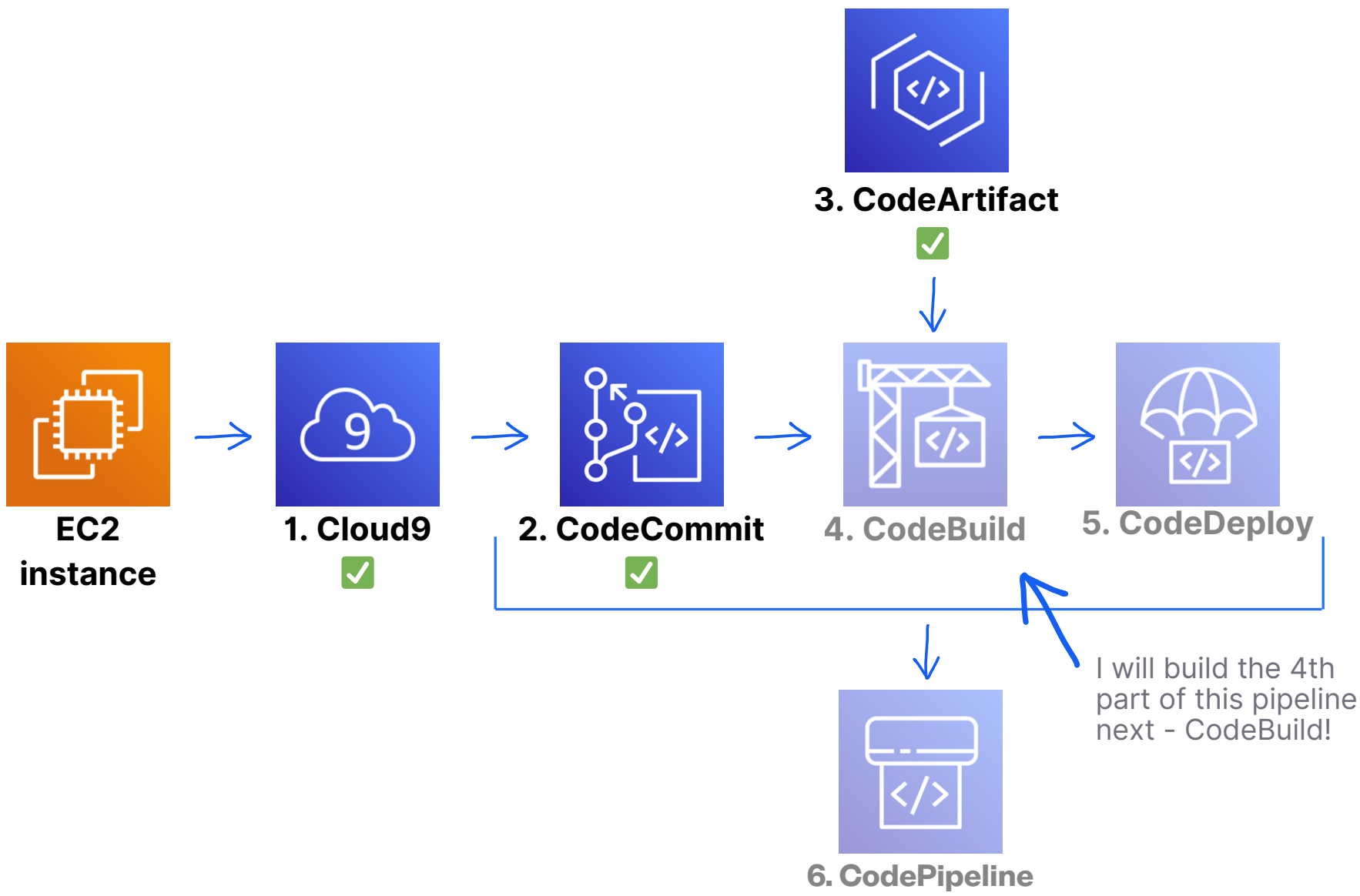
Kanika Mathur
github.com/KanikaGenesis

My CI/CD pipeline so far...

1. Cloud9 is responsible for IDE that allows you to write ,run and debug your code.

2. CodeCommit is responsible for hosting git repositories.

3. CodeArtifact is responsible for securely storing and sharing packages/dependencies used in software development process.



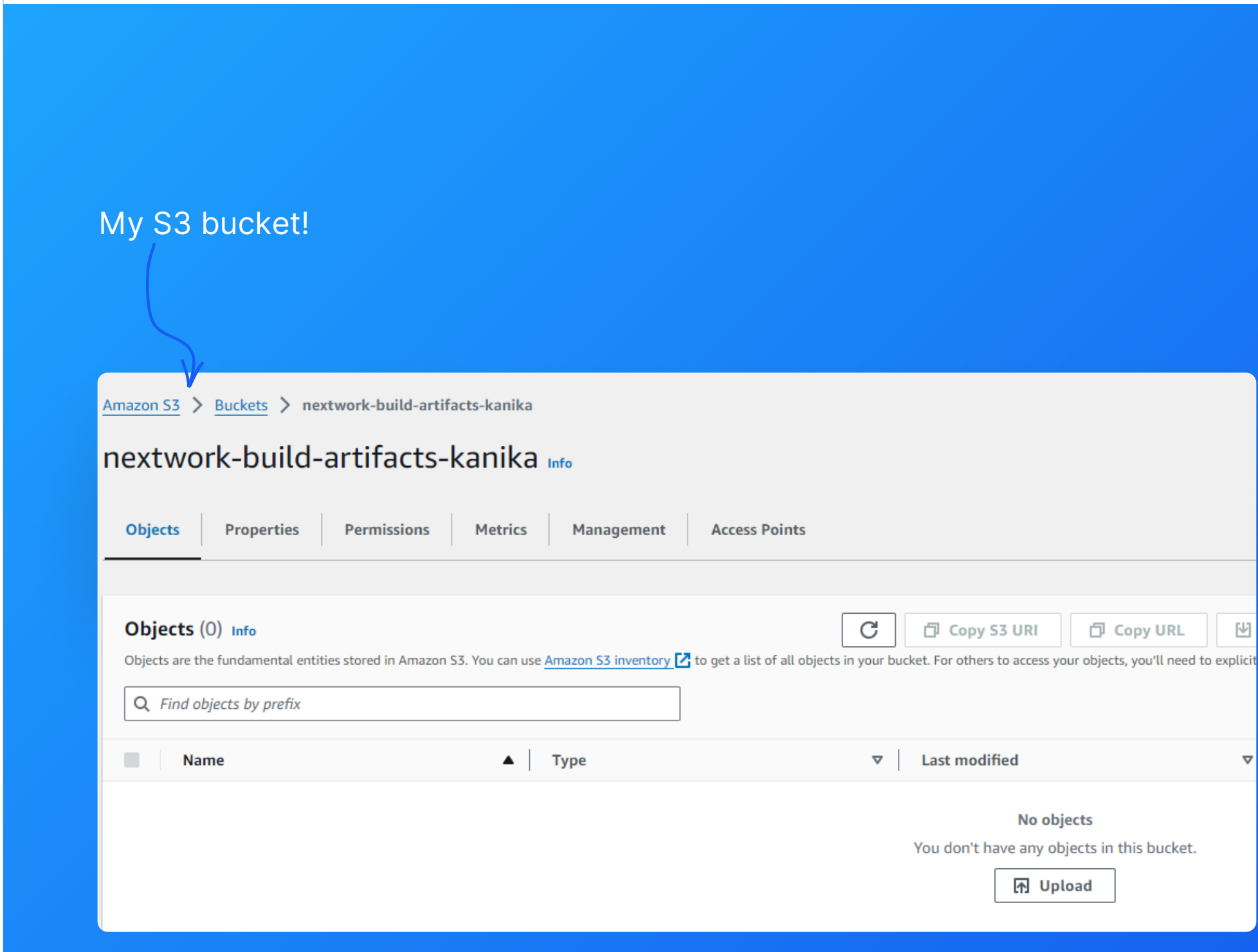

Kanika Mathur
github.com/KanikaGenesis

Set up an S3 bucket

- I started my project by creating an S3 bucket because this bucket will later catch an important artifact that gets created from the build process I am setting up with CodeBuild.

- The key artifact that this S3 bucket will capture is called a WAR (Web Application Resource) file.

- This file is important because it ensure that any server that will host my web app will have all of the resource/tools it needs to successfully run my application.



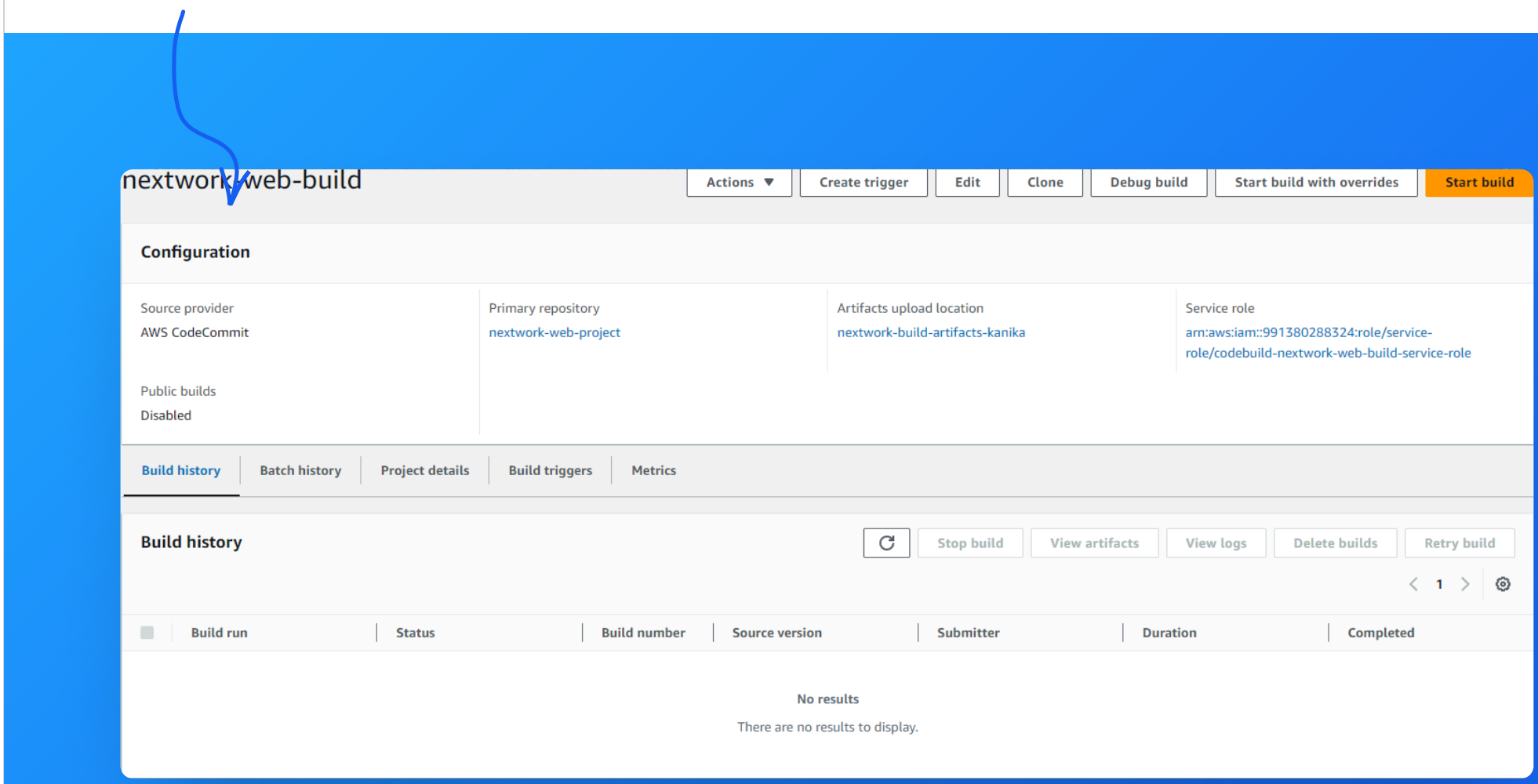

Kanika Mathur
github.com/KanikaGenesis

Set up a CodeBuild project

When creating a project in CodeBuild, there were 5 key configurations I set up:

1. **Source**, which means the source of the project's code/files. I selected AWS CodeCommit, as that is my cloud repository for my project's files.
2. **Environment**, which means the set of resources/configurations required to build my web app. I selected an EC2 instance running Amazon Linux 2 and Java Coretto 8 as the runtime.
3. **Buildspec**, which are the set of commands to run in my local computer's terminal during the build process. I selected "use a buildspec file."
4. **Artifacts**, which means where CodeBuild should store the build artifacts that get produced during the build process. I selected the S3 bucket I created.
5. **Logs**, which means whether I'd like to keep records of every single build process that CodeBuild does. I enabled CloudWatch Logs.

My completed project ready for the first build!




Kanika Mathur
github.com/KanikaGenesis

Create a buildspec.yml file

- I created a buildspec.yml file at the root of our code repository.
- This file contains four phases that tells our build environment what commands to run. These four phases are:

1. install -i.e. install these dependencies before you start compiling (in this project, it's Java Coretto 8).
2. pre_build - i.e. before Maven starts building retrieve an access token to our CodeArtifact repository so we can fetch dependencies later.
3. build - i.e. to build our web app project, run these commands.
4. post_build - i.e. after building our web app project, package up the machine code using settings in the setting.xml file.

A peek into my buildspec.yml

Welcome | index.jsp | settings.xml | buildspec.y

```yaml
version: 0.2

phases:
  install:
    runtime-versions:
      java: corretto8
  pre_build:
    commands:
      - echo Initializing environment
      - export CODEARTIFACT_AUTH_TOKEN=`aws codeartifact get-authorization-token --domain ne
  build:
    commands:
      - echo Build started on `date`
      - mvn -s settings.xml compile
  post_build:
    commands:
      - echo Build completed on `date`
      - mvn -s settings.xml package
artifacts:
  files:
    - target/nextwork-web-project.war
  discard-paths: no
```

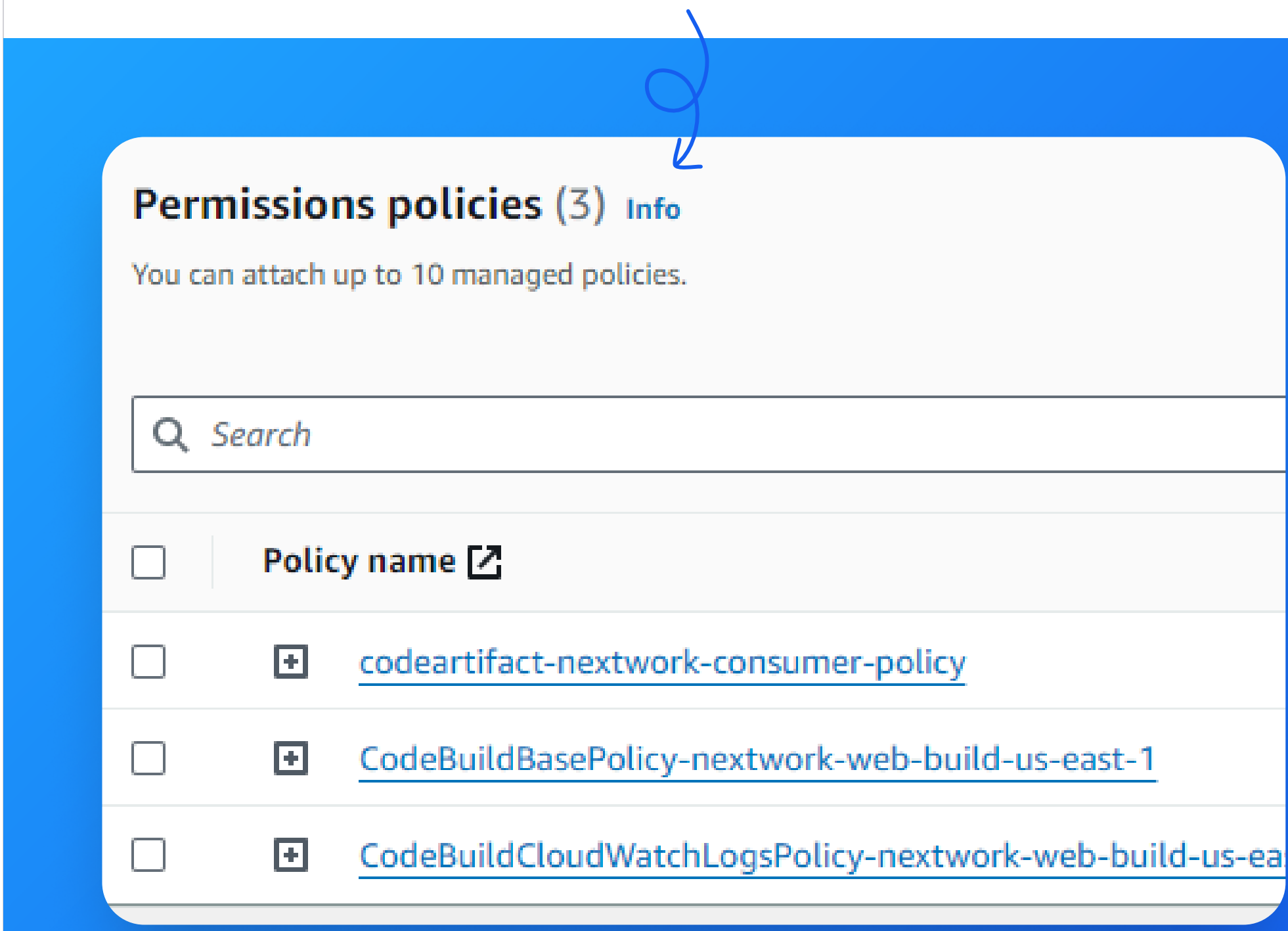
Kanika Mathur
github.com/KanikaGenesis

Edit CodeBuild's IAM role

- Before I start building my web app project (exciting!), I modified my CodeBuild project's service role first. This role was first created when I set up my CodeBuild project (I checked the setting for create new service role).

- I attached a new policy called **codeartifact-nextwork-consumer-policy** to my CodeBuild project's IAM role. This means my CodeBuild project now has access to the packages/dependencies that it will later compile.

Updating permission policies for my CodeBuild project's IAM role.




Kanika Mathur
github.com/KanikaGenesis

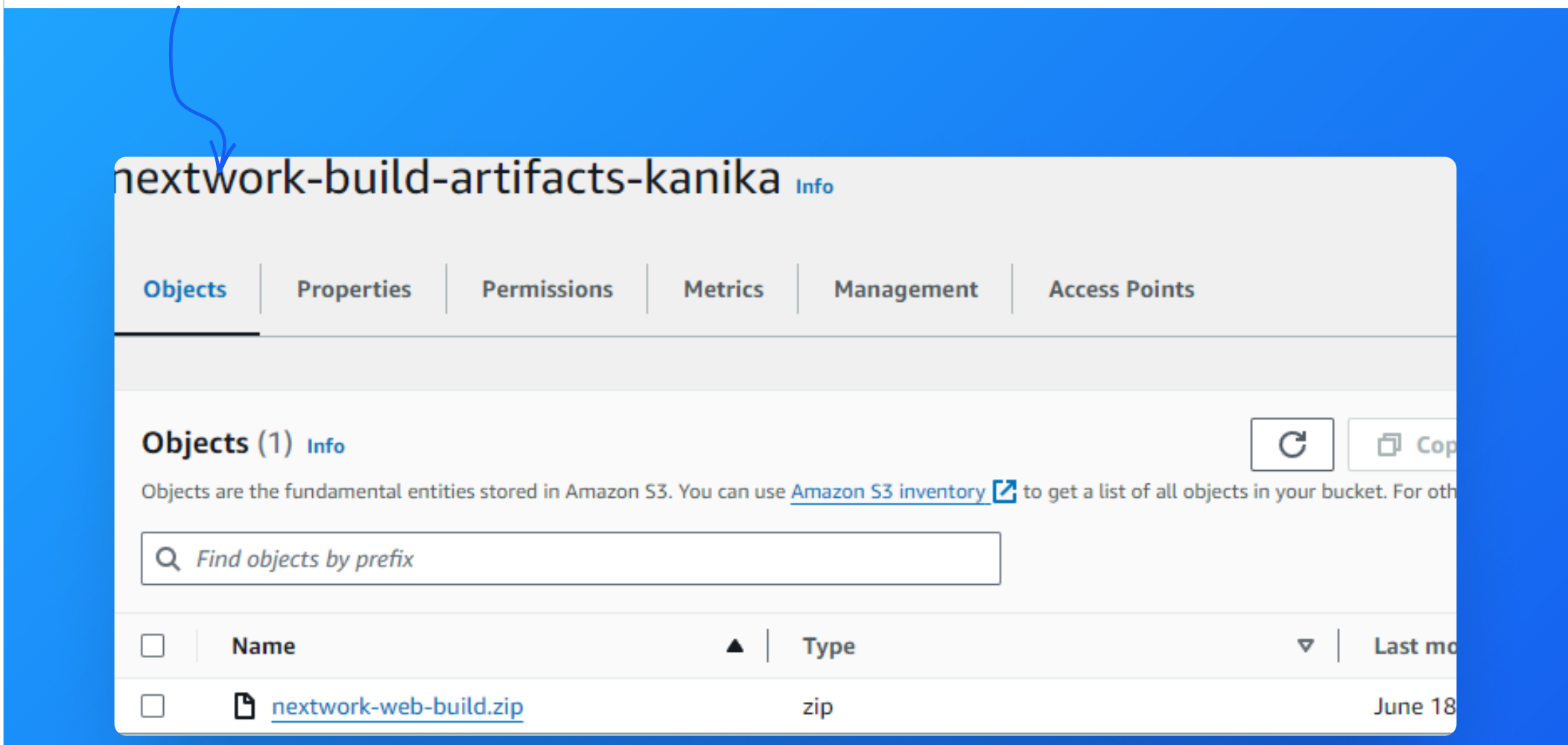
My first project build 💪

- To build my project, all I had to do was select the Start build button in my CodeBuild console - super simple!

- The build process in CodeBuild took about 5 minutes.

- Once the build is complete, I checked our S3 bucket was set up at the start of this project to catch the build artifacts that get produced from our build process. I saw the zip file i.e. the WAR file (a bundle up package of all files/resources a server will need to run my web app), which verified that the build was completed successfully.

My completed project ready for the first build!




Kanika Mathur
github.com/KanikaGenesis

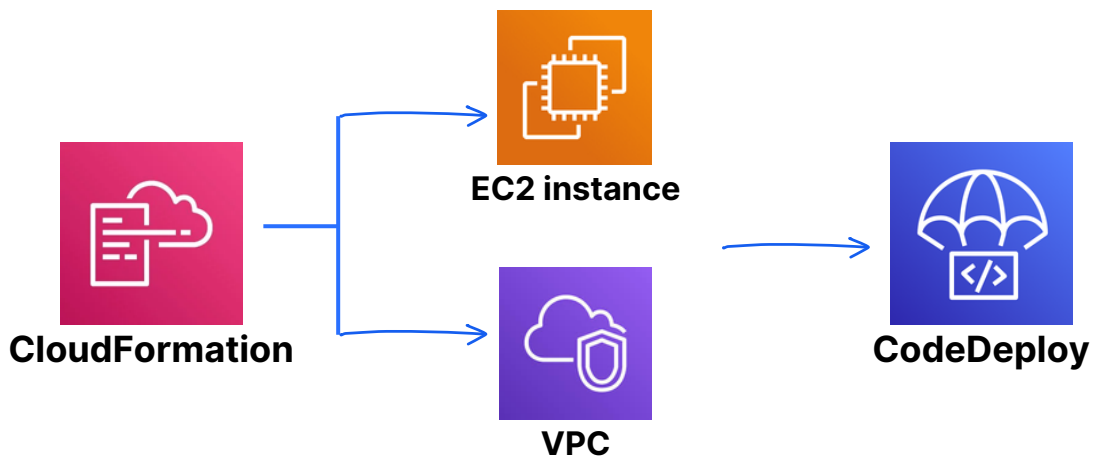
Set up an EC2 instance

- I set up an EC2 instance and VPC because deploying my web app requires a separate environment (my production environment).

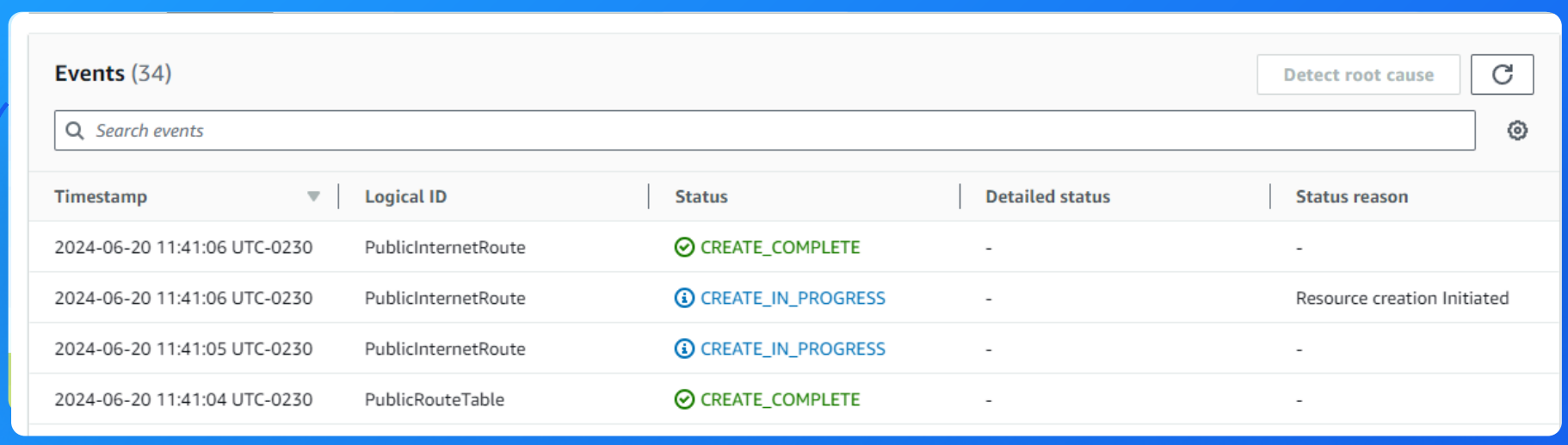
- The production environment is different to the build environment because deploying my web app will require different tools/dependencies in order to function properly. It is also best practice to separate environments for developers to test code without interfering with what users see.

- To set up my EC2 instance and VPC, I used AWS CloudFormation.

- The diagram below illustrates the flow of using CloudFormation to create two new resources (EC2 instance, VPC) that will run/deploy my web app (which we will see in action when we set up AWS CodeDeploy).



A peek into my CloudFormation stack's deployment!




Kanika Mathur
github.com/KanikaGenesis

Write bash scripts

- Scripts are collections of commands in a file i.e when you run a script, you are telling the terminal to run (line by line) what you've written in the script.

- Bash is a specific type of script language.

- I created 3 scripts for this the deployment process:
- install_dependencies.sh installs Tomcat and HTTPD (web servers)
- start_server.sh starts up our web servers
- stop_server.sh stops our web servers
    a.

A peek into appspec.yml!

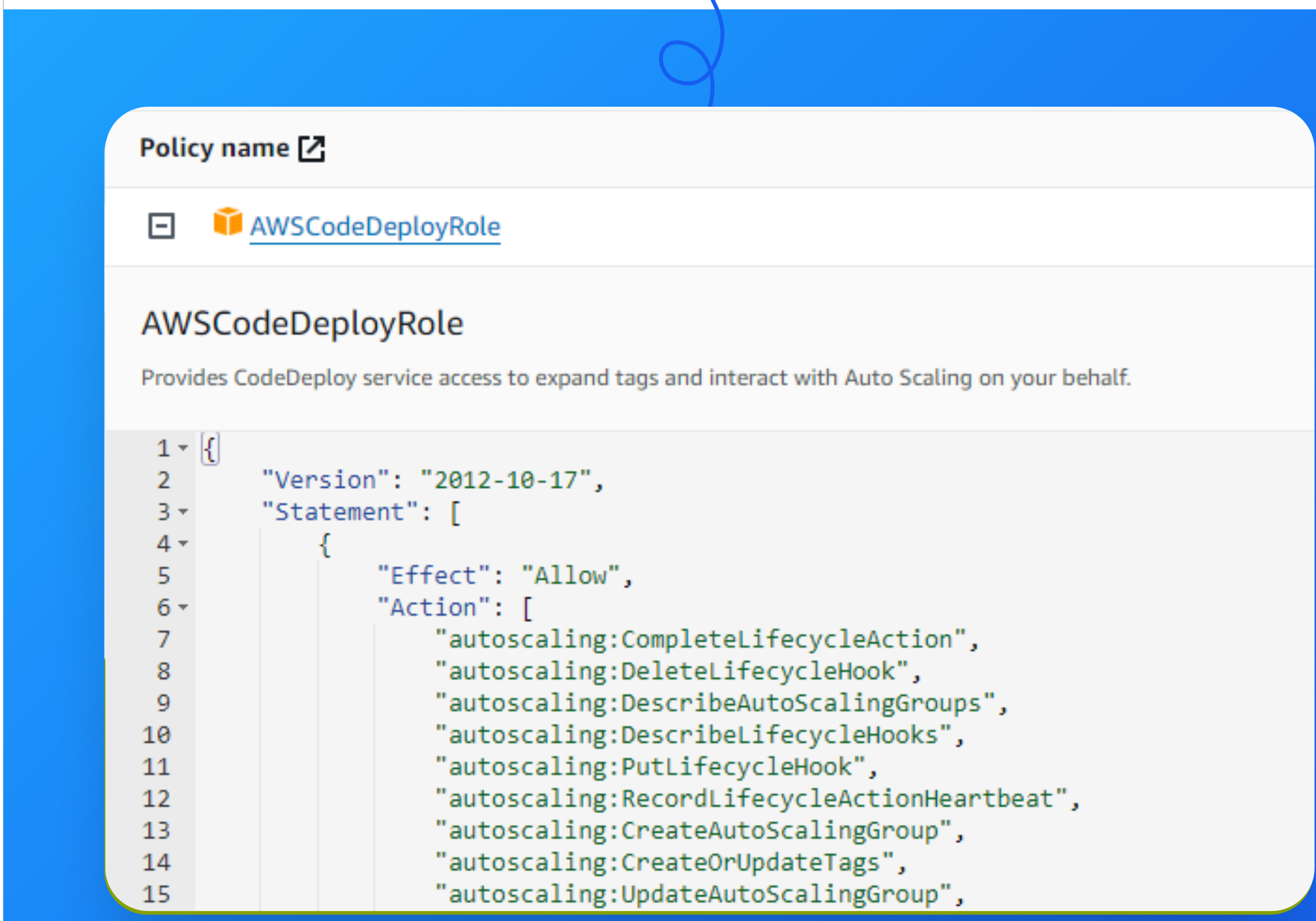
```
version: 0.0
os: linux
files:
  - source: /target/nextwork-web-project.war
    destination: /usr/share/tomcat/webapps/
hooks:
  BeforeInstall:
    - location: scripts/install_dependencies.sh
      timeout: 300
      runas: root
  ApplicationStart:
    - location: scripts/start_server.sh
      timeout: 300
      runas: root
  ApplicationStop:
    - location: scripts/stop_server.sh
      timeout: 300
      runas: root
```

Kanika Mathur
github.com/KanikaGenesis

CodeDeploy’s IAM Role

- I created an IAM service role for CodeDeploy because CodeDeploy needs access to other AWS servvices e.g. EC2 in order to successfully deploy web app.
- To set up CodeDeploy IAM role, I used an AWS Managed Policy called AWSCodeDeployRole which automatically adds default permissions that CodeDeploy often needs.

The permissions granted by my CodeDeploy IAM role.




Kanika Mathur
github.com/KanikaGenesis

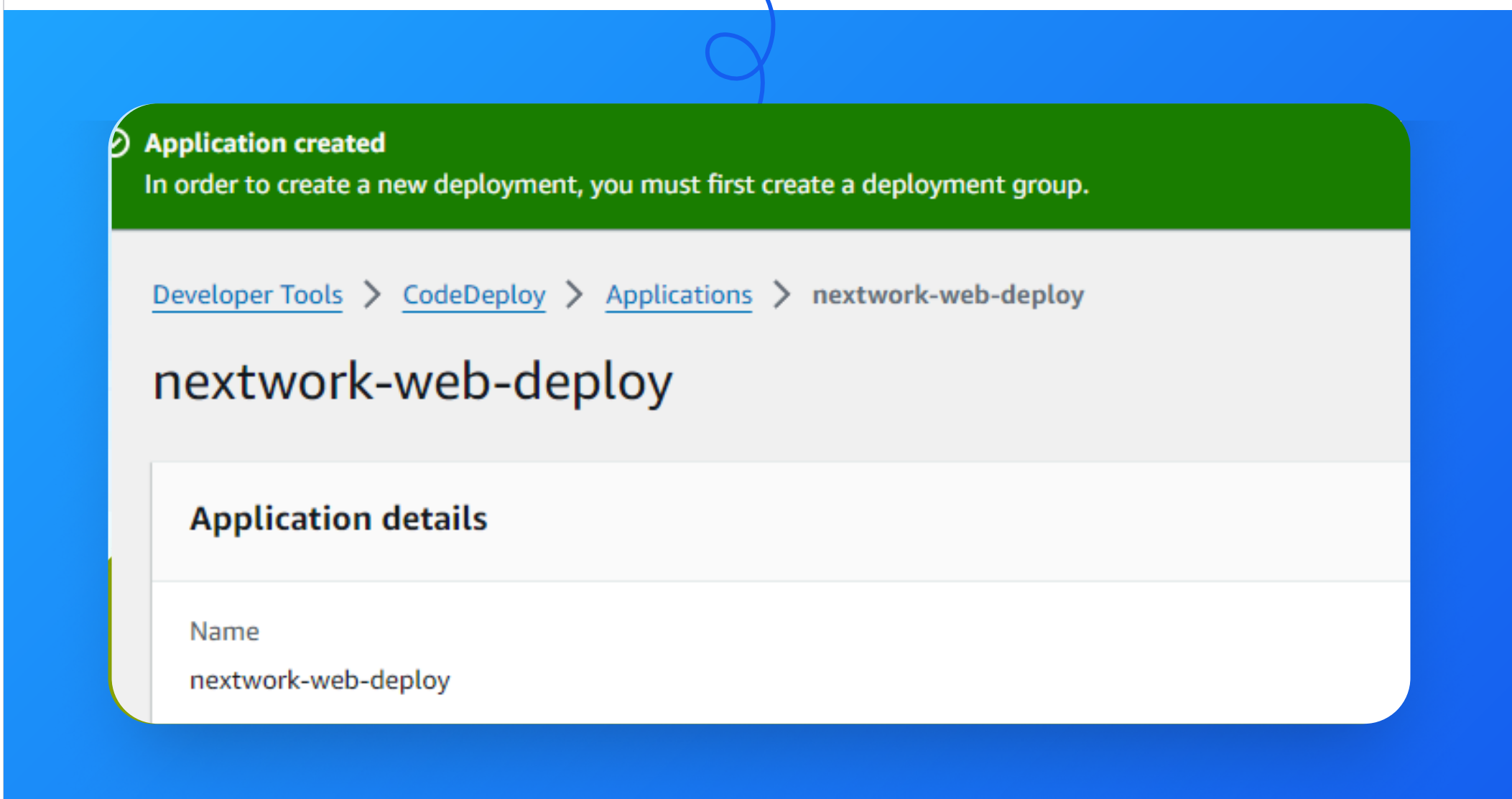
Create a CodeDeploy app

- A CodeDeploy application means a saved configuration/setup template on how to deploy my web app.

- To create a CodeDeploy application, I had to select a compute platform, which determines the environment in which my web application will be hosted. With every platform comes its own set of requirement for deploying a web app.

- The compute platform I chose was EC2, because I have used an EC2 instance as my web server.vChoosing EC2 gives me the most amount of control out of all the three compute platform options. It's also great for my learning as I will be exposed to more configuration options!

My CodeDeploy app ready for a deployment!




Kanika Mathur
github.com/KanikaGenesis

Set up a deployment group

- A deployment group means the configuration set for a specific deployment scenario.

- To create my deployment group, I set up a:
  - **Service role**, which is an IAM role that I am using for mu deployment group to give CodeDeploy access to my Web Server EC2 instance.

  - **Deployment type**, which is how deployment will be managed. I chose In place, i.e. my Web Server EC2 instance will deploy the latest web app right away without needing to create a new environment.

  - **Environment configuration**, which means the type of servers that will be used to deploy my web app - in this case, I just used Amazon EC2 instances (without needing Auto Scaling groups).

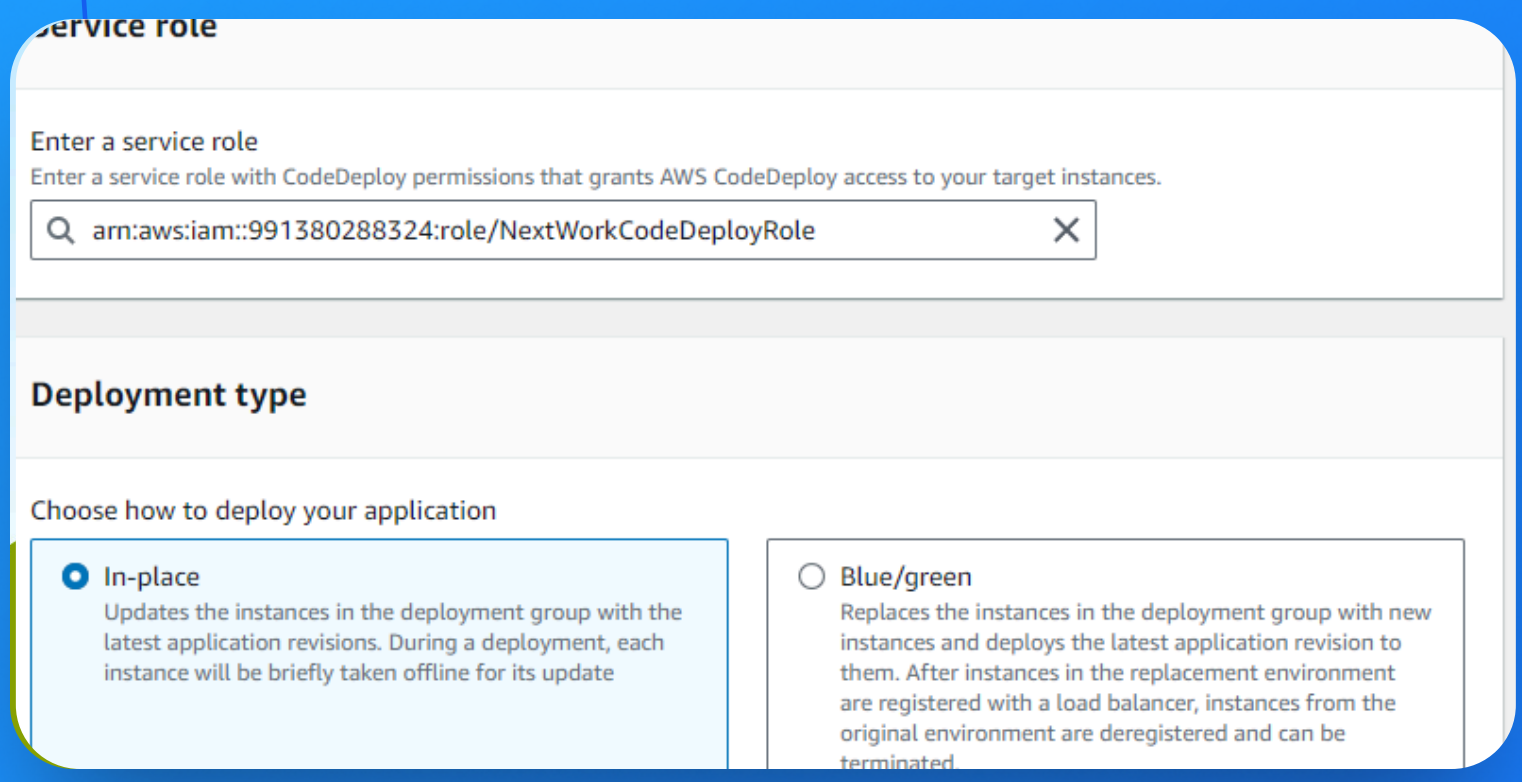
  - **CodeDeploy Agent**, which is CodeDeploy's helper for communicating with my EC2 Web Server and making sure that carry out the instructions for deployment.

  - **Deployment settings**, which are whether deployment will be staggered out or done all out once.

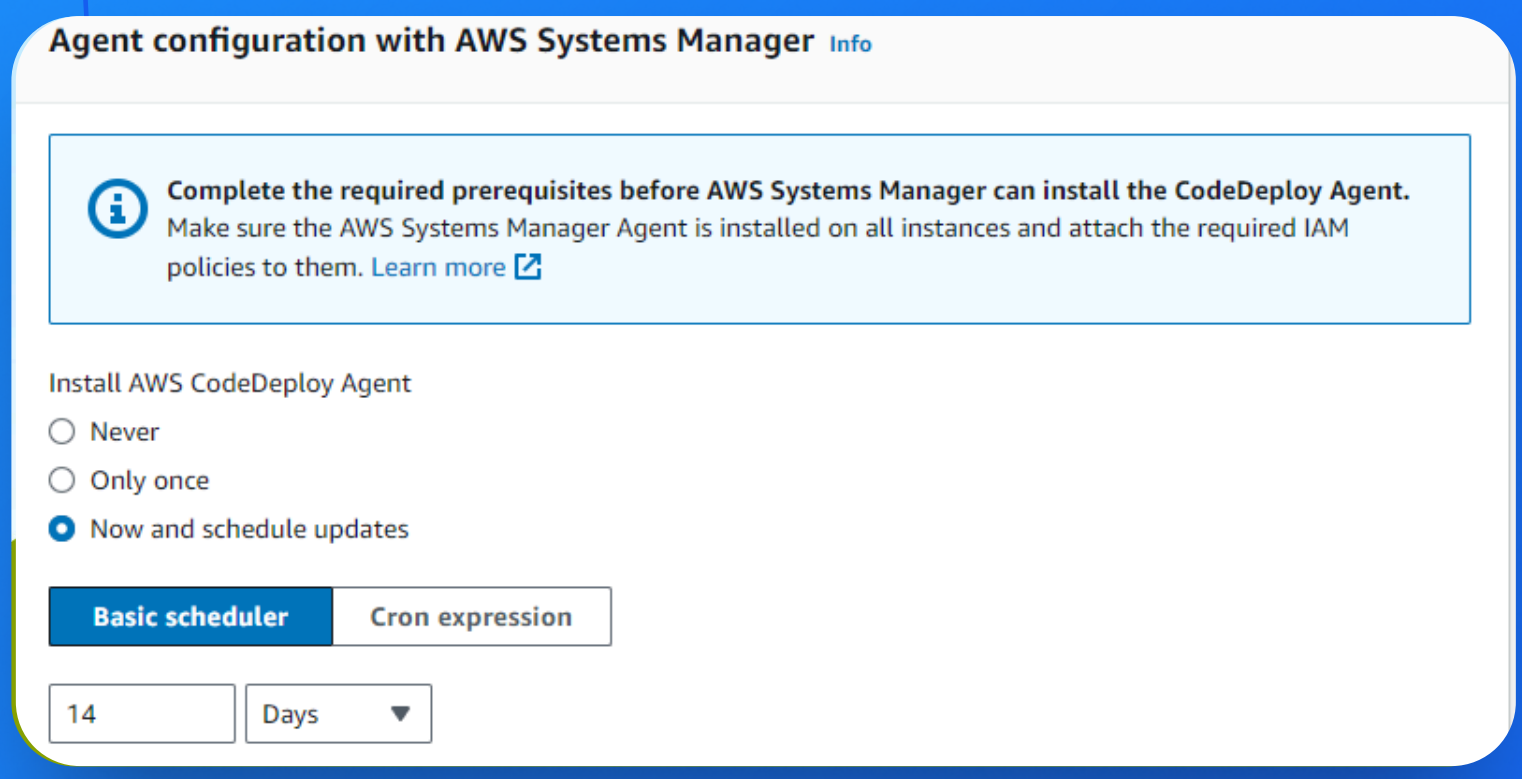
- For the load balancer setting, I disabled the load balancer.
  - This was because I only have one web server deploying my web app.

Kanika Mathur
github.com/KanikaGenesis

Setting up the service role + deployment type.



Setting up the CodeDeploy Agent.



Kanika Mathur
github.com/KanikaGenesis

Deployment success! 🚀

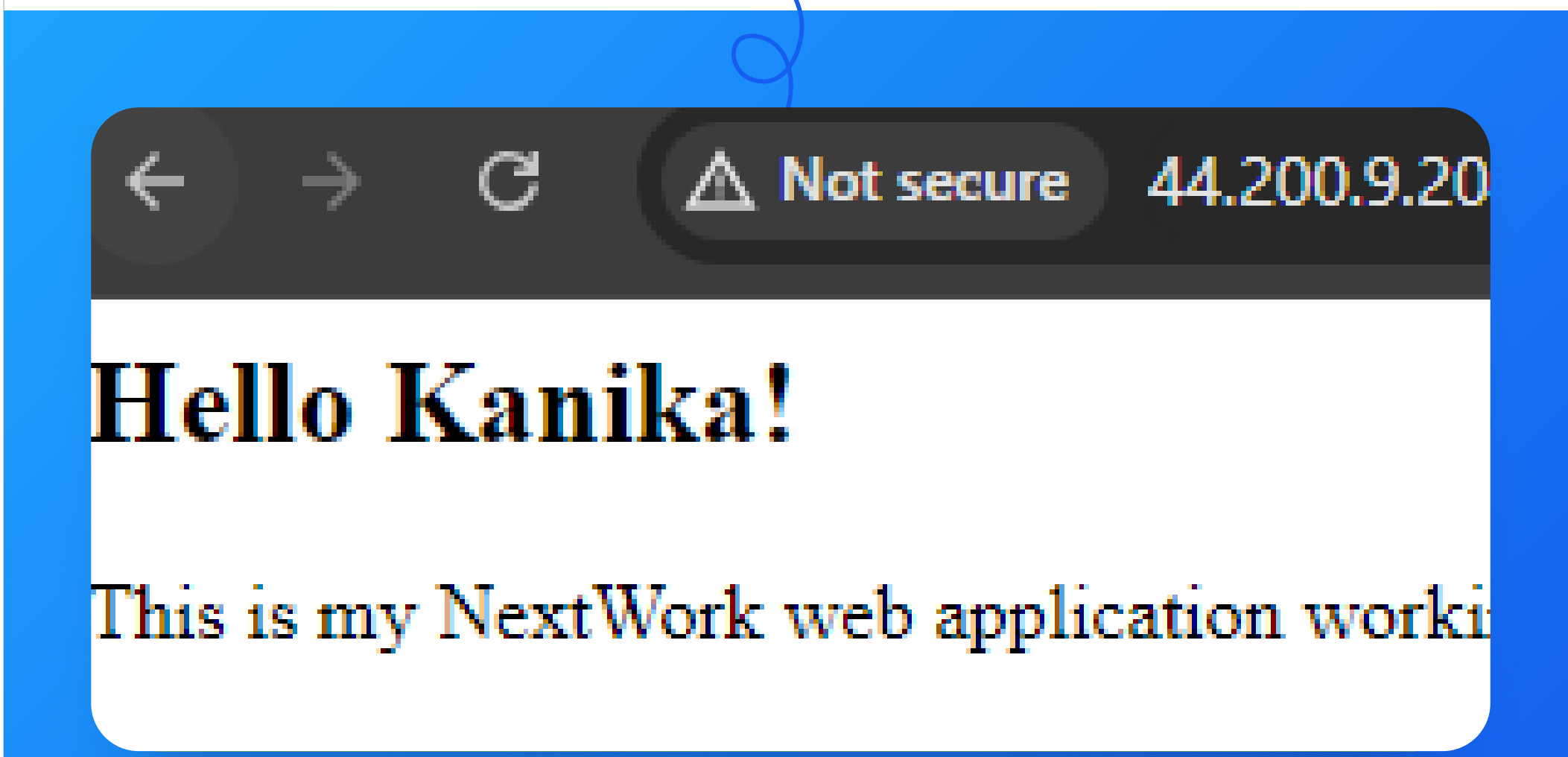
- After setting up my deployment group, I was ready to kick off my first deployment!

- To create my deployment, I had to set up a revision location, which means the location of my web app's build artifacts.

- My revision location was the Java WAR file/zip file containing my compiled web app code. It was sitting inside my S3 build artifacts bucket.

- To visit my web app, I had to visit my Web Server which was in my EC2 console. I have to find my EC2 instance's IPv4 address.

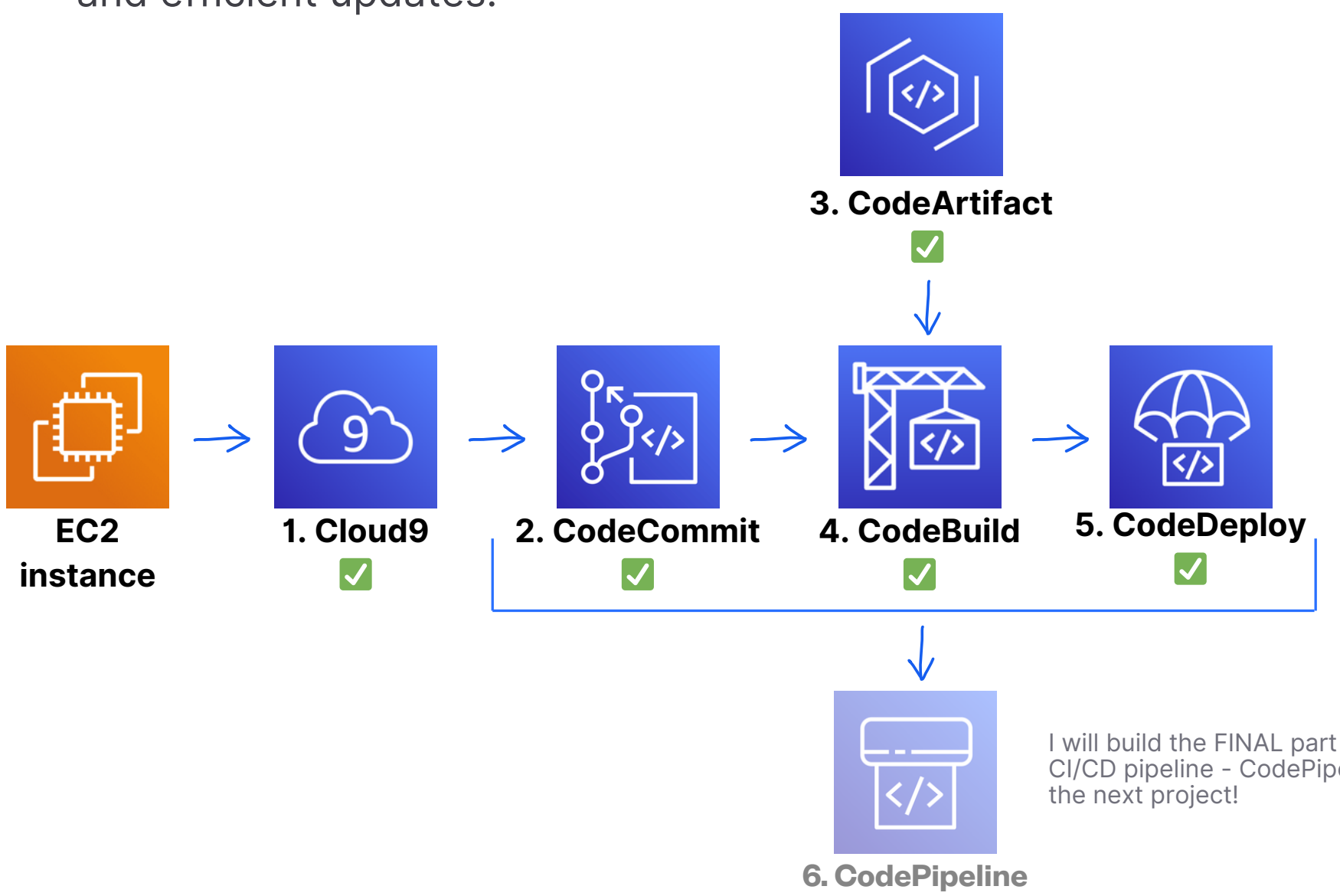
Amazing! I could see my web app taking shape.




Kanika Mathur
github.com/KanikaGenesis

My CI/CD pipeline so far...

- **AWS Cloud9** is responsible for providing a cloud-based IDE where I can write, run, and debug my code.
- **AWS CodeCommit** is responsible for securely hosting my Git repositories, enabling version control and collaboration.
- **AWS CodeArtifact** is responsible for managing and storing software packages, making it easy to share dependencies across teams.
- **AWS CodeBuild** is responsible for compiling my source code, running tests, and producing build artifacts.
- **AWS CodeDeploy** is responsible for automating the deployment of my applications to various compute services, ensuring smooth and efficient updates.




kanika Mathur
github.com/KanikaGenesis

Set Up a CI/CD Pipeline

- A CI/CD pipeline is a practice of continuous integration, continuous deployment. This makes sure that changes made to source code is constantly being integrated/updated in a shared Git repository, so that other developers working from the same code base are always the latest version of the code (continuous integration). It also makes sure that end users are always seeing the latest version/updates to an application (continuous deployment).

To set up a CI/CD in CodePipeline, I configured three stages:

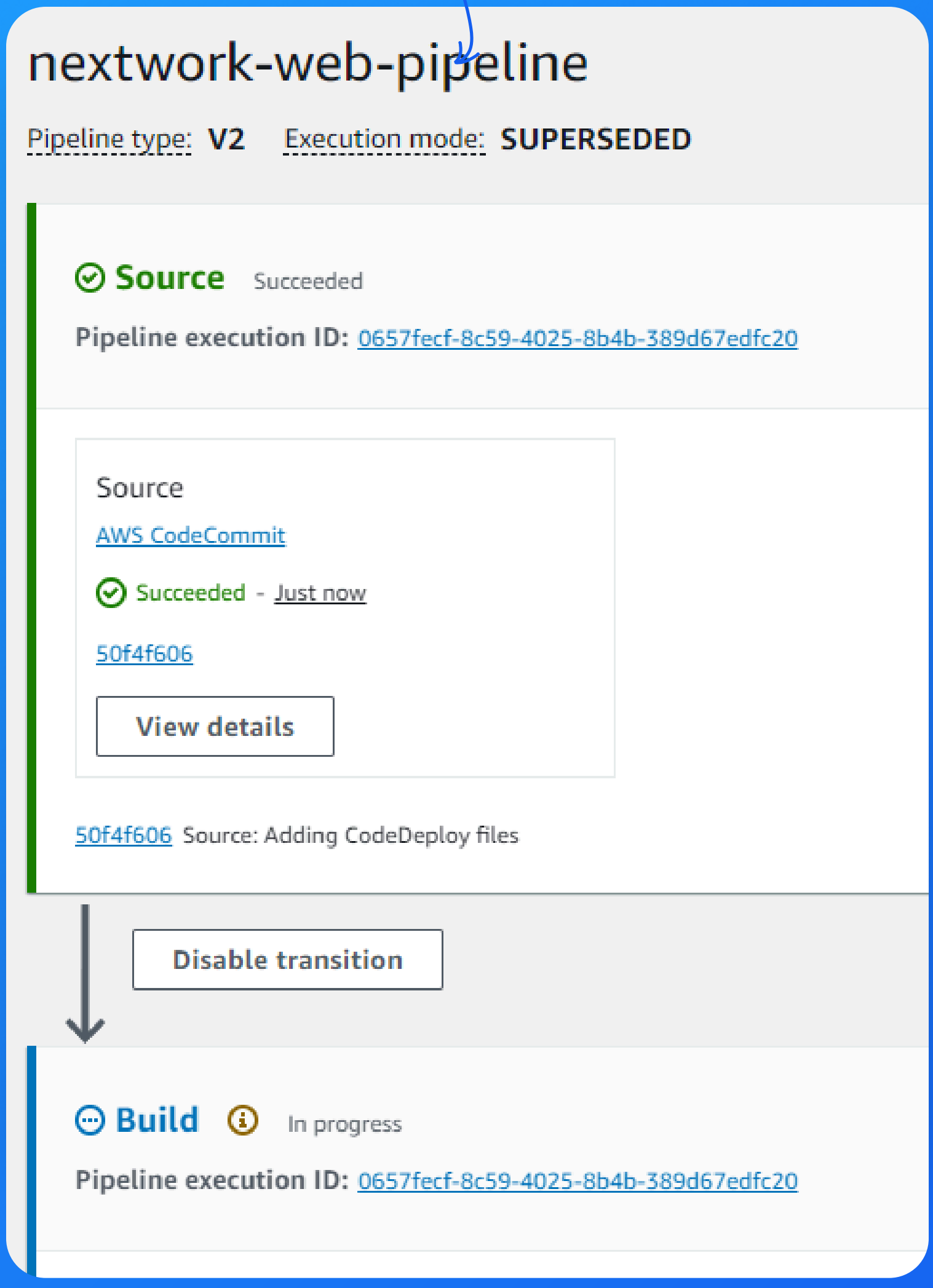
- The **source** stage, which means the source code for my web app. This is currently stored in CodeCommit.

- The **build** stage, which means the service that is managing the build process for my web app (i.e. compiling and packaging my web app into a handy WAR file.). This CodeBuild.

- The **deploy** stage, which means the service that is managing the deployment process for my web app (i.e. making my web app available to the world). This is CodeDeploy.

kanika Mathur
github.com/KanikaGenesis

A peek into my pipeline set up in CodePipeline!



kanika Mathur
github.com/KanikaGenesis

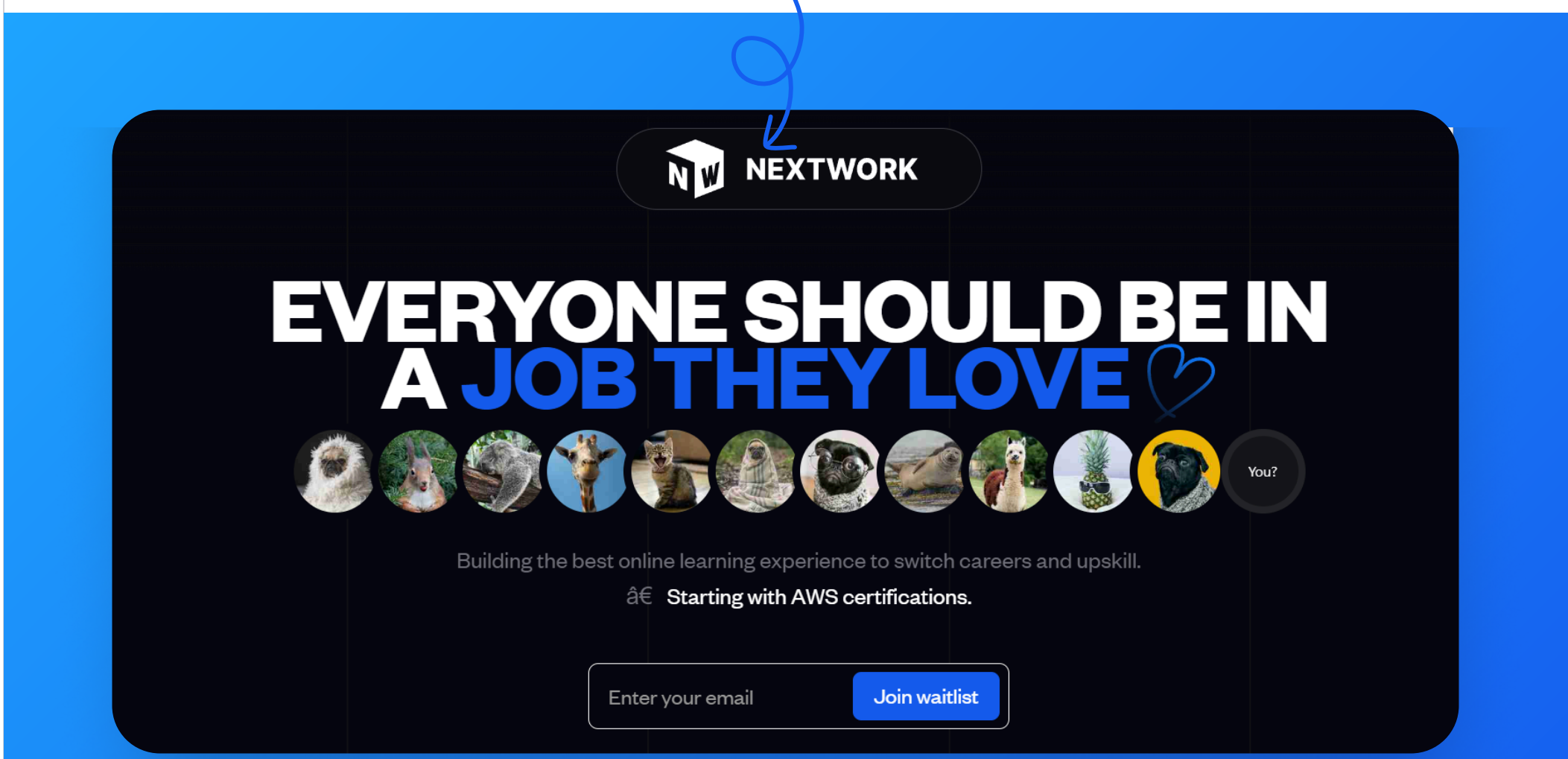
Release a Change

- My CI/CD pipeline gets triggered by a commit in my local working environment (Cloud9) - which updates my CodeCommit repository.

- I tested this by making **two** updates to my web app's source code. These changes were an edit to **index.jsp** and uploading a folder of image assets in the **webapp** folder of my project files.

- Once my pipeline executed successfully, I checked the IPv4 address of my web server that is hosting my web app.

My web app automatically updated after I committed a change.




kanika Mathur
github.com/KanikaGenesis

Trigger A Rollback

- A rollback in a pipeline means reverting the version of our code that a stage of our pipeline is referring to.

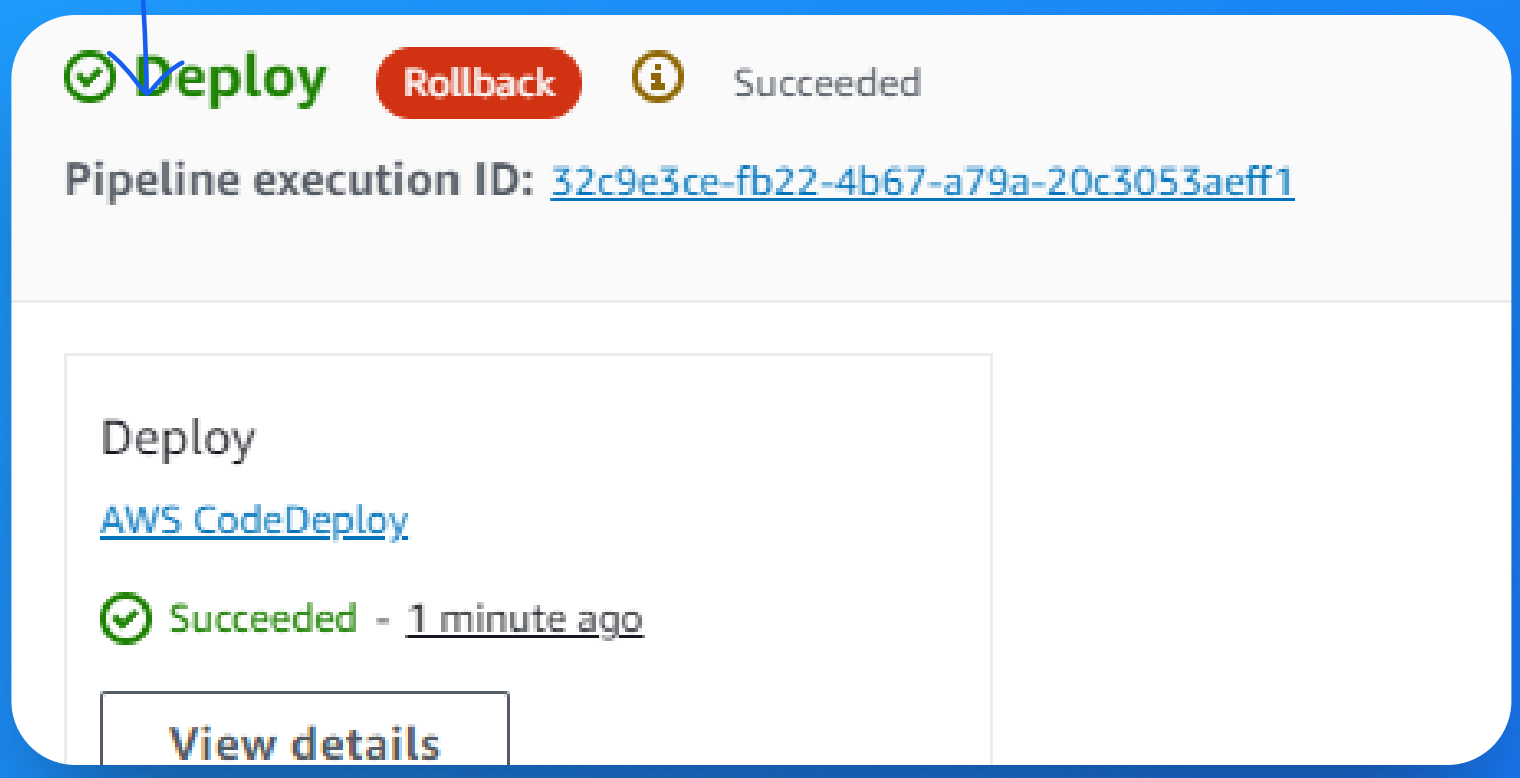
- I initiated a rollback on the deploy stage. I checked the source stage, and learnt that the source stage was unaffected by the rollback, and stayed using the latest version of my source code.

- Once the rollback was complete, my web app's appearance reverted back to reflect the simple version of my index.jsp file before I ever updated my index.jsp file and uploaded new image assets.

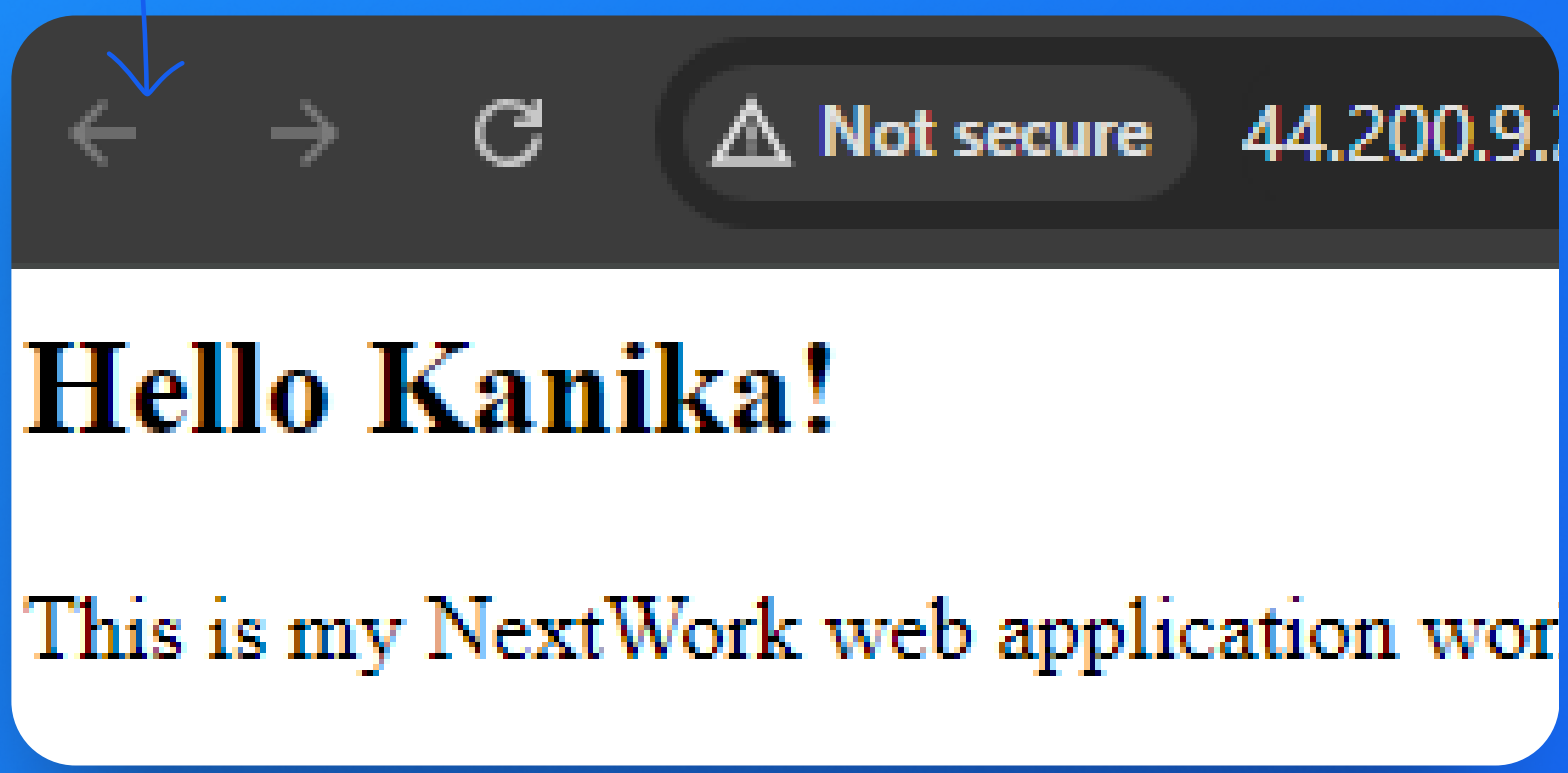
- To update my Deploy stage back to the latest version of my source code, I **released** a change in my CodePipleline pipleline. Releasing a change means the latest version of my source code gets referenced across the Source, Build AND Deploy stages of my pipeline.

kanika Mathur
github.com/KanikaGenesis

My CodeDeploy panel showing a successful rollback.



My web app returned to its previous state after the rollback.




kanika Mathur
github.com/KanikaGenesis

My CI/CD Pipeline

1. **AWS Cloud9** is responsible for providing an integrated development environment (IDE) for writing, running, and debugging code in the cloud.

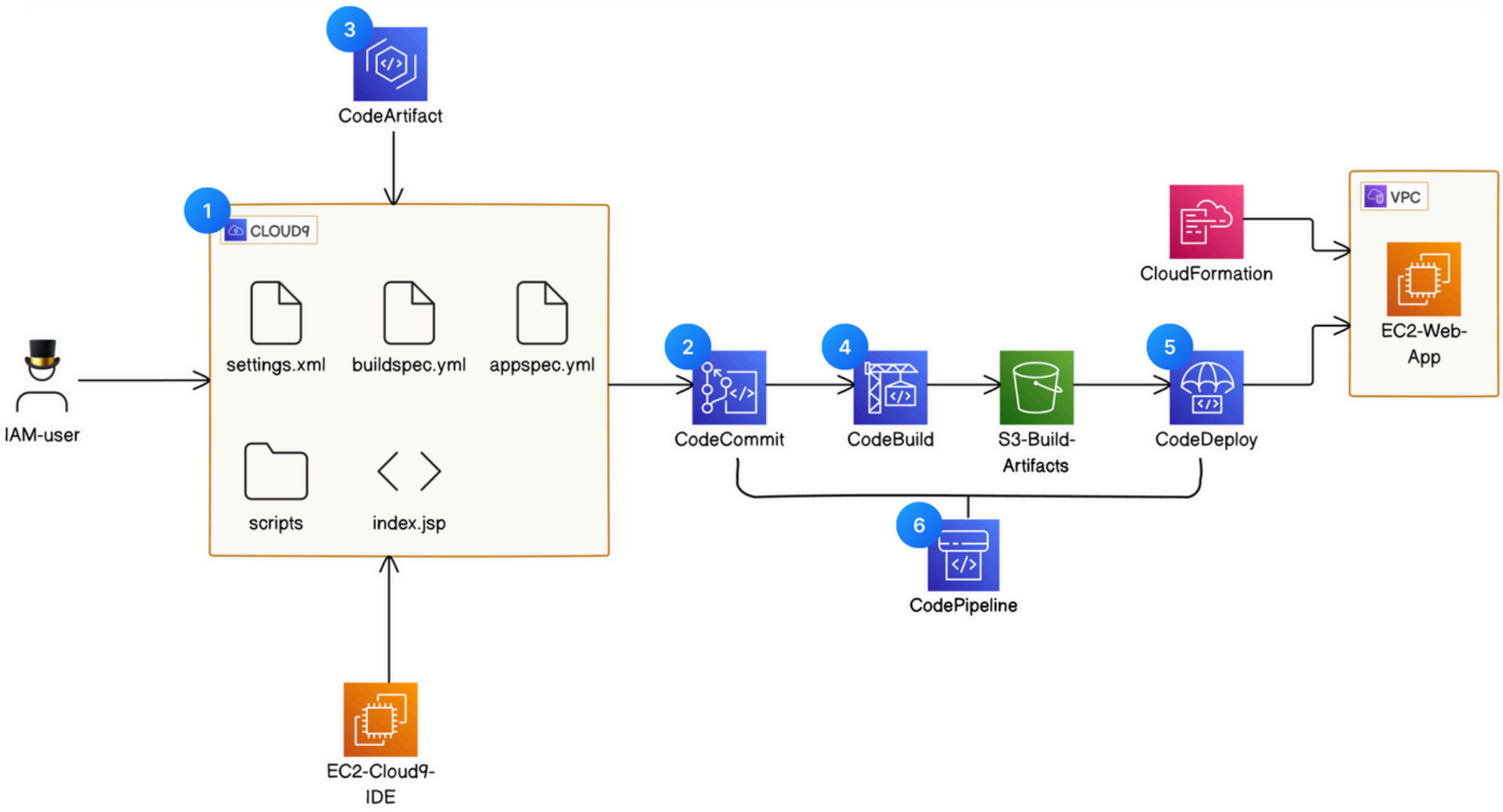
2. **AWS CodeCommit** is responsible for hosting secure Git repositories to store and version control the source code.

3. **AWS CodeArtifact** is responsible for managing software packages and dependencies, making it easy to store, publish, and share them.

4. **AWS CodeBuild** is responsible for compiling and packaging my app's source code into a handy WAR file.

5. **AWS CodeDeploy** is responsible for deploying that WAR file on servers, so users can see my web app!

6. **AWS CodePipeline** is responsible for automating the integrations between CodeCommit, CodeBuild and CodeDeploy.




kanika Mathur
github.com/KanikaGenesis

My key learnings

1. Through this Cloud DevOps project, I gained a solid understanding of core principles such as **continuous integration**, **continuous delivery/deployment**, and **infrastructure as code**. Proficiency in using **AWS Cloud9** as an integrated development environment was a crucial skill acquired, enabling effective coding and debugging.

2. Mastering **Infrastructure as Code (IaC)** using **AWS CloudFormation** allowed me to define and provision infrastructure resources efficiently. This approach emphasized the benefits of version-controlled infrastructure and repeatable deployments, ensuring consistency and reliability. Implementing a **CI/CD** pipeline with **AWS CodePipeline**, **CodeBuild**, and **CodeDeploy** automated the build, test, and deployment processes, significantly improving the release cycle's speed and reliability.

3. Integrating various AWS services such as **S3**, **EC2**, and **IAM** into the DevOps workflow provided hands-on experience in configuring permissions and managing access control. This integration allowed for seamless data storage, compute resource management, and secure user authentication.

4. Throughout the project, I faced and overcame challenges related to environment configuration, service integrations, and pipeline automation. These experiences enhanced my problem-solving skills and underscored the importance of best practices like automated testing, incremental deployments, and continuous feedback loops. The project has prepared me to apply the knowledge and skills acquired to real-world scenarios and future projects.

NEXTWORK

Everyone should be in a job they love. yes!

Check out community.nextwork.org for more free projects