**AWS CI/CD Project Overview**

Welcome to this video series on creating a CI/CD pipeline using AWS services. Previously, we explored continuous delivery using Jenkins. However, there are numerous ways to achieve continuous delivery based on your specific requirements. In this project, we will focus on using AWS-managed services exclusively, avoiding tools like Git, Jenkins, and EC2 instances.

**Project Goals and Components**

Our primary objective is to integrate various AWS services to create a complete CI/CD pipeline. Here’s a breakdown of the services we will use and their roles in the project:

1. **AWS CodeCommit**: This service will be used to store our source code instead of GitHub.
2. **AWS CodeBuild**: This service will build our artifacts, replacing Jenkins' build functionality. We can set up multiple CodeBuild projects for different tasks such as:
   * Running code analysis
   * Building the project
   * Running unit tests
3. **AWS S3**: Once the artifact is built, it will be stored in an S3 bucket.
4. **AWS CodeDeploy**: This service will deploy the artifacts to various targets. In our case, we will use AWS Elastic Beanstalk.
5. **AWS Elastic Beanstalk**: We will deploy our Tomcat application on Elastic Beanstalk. This platform will host our web application.
6. **AWS RDS**: Our application requires a MySQL database, which we will provision using Amazon RDS.
7. **AWS CodePipeline**: This service will combine all the steps into a seamless pipeline. It will trigger the entire process whenever there is a code change.

**Project Workflow**

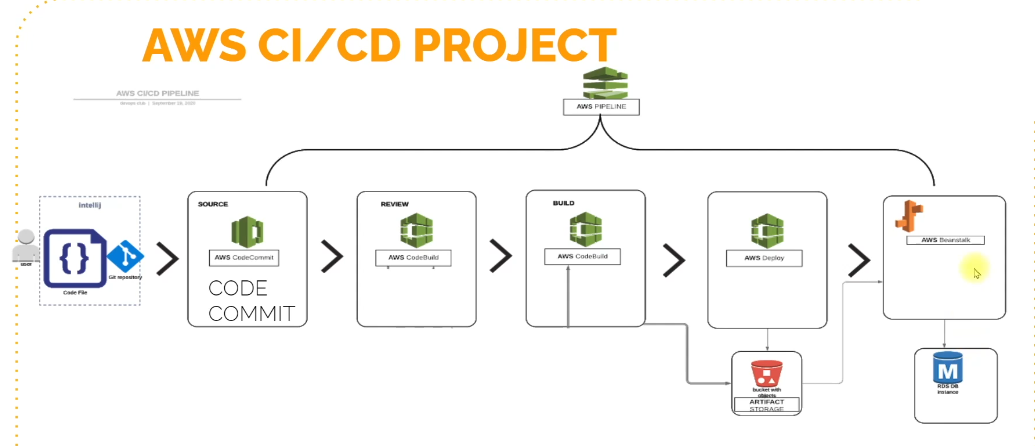
Here’s the step-by-step workflow we will follow:

1. **Setup Infrastructure**:
   * Create and configure an Elastic Beanstalk environment with a Tomcat platform.
   * Set up an RDS instance for the MySQL database and connect it to Elastic Beanstalk.
2. **Configure CI/CD Pipeline**:
   * Set up CodeCommit to store the source code.
   * Configure CodeBuild to build the project and run tests.
   * Use S3 to store the built artifacts.
   * Set up CodeDeploy to deploy the artifacts to Elastic Beanstalk.
   * Integrate everything using CodePipeline, ensuring that every code change triggers the pipeline, resulting in a deployed application.

**Let's Get Started**

In the upcoming videos, we will dive into each step, starting with setting up our infrastructure. We'll create and configure our Elastic Beanstalk and RDS environments and then move on to setting up the CI/CD pipeline. By the end of this series, you will have a fully automated CI/CD pipeline running entirely on AWS services.

Stay tuned and let’s get into action now!



### AWS Elastic Beanstalk Overview

Welcome to this video where we will dive into AWS Elastic Beanstalk, a Platform as a Service (PaaS) that simplifies running application workloads. Elastic Beanstalk abstracts away much of the infrastructure management involved in deploying applications, allowing you to focus on writing code.

#### Key Features of AWS Elastic Beanstalk:

1. **Ease of Use**: Deploy applications quickly without setting up EC2 instances, load balancers, auto-scaling groups, or monitoring tools.
2. **Supports Multiple Environments**: Create development, QA, staging, and production environments easily.
3. **Automated Management**: Automatically handles the provisioning, load balancing, auto-scaling, and health monitoring of your applications.
4. **Wide Platform Support**: Supports multiple platforms including Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker.
5. **No Additional Cost**: Only pay for the AWS resources you use, such as EC2 instances and load balancers.

### Steps to Set Up an Elastic Beanstalk Environment

#### Step 1: Create an Elastic Beanstalk Application

1. **Navigate to Elastic Beanstalk**:
   * Go to the AWS Management Console and select Elastic Beanstalk.
2. **Create Application**:
   * Click on "Create Application".
   * Provide an application name (e.g., vprofile).
   * Optionally, add tags for better resource management.

#### Step 2: Create an Environment

1. **Create Environment**:
   * Click on "Create environment".
   * Select "Web server environment".
   * Provide a name for your environment (e.g., vprofile-prod).
   * Ensure the domain name is unique.
2. **Select Platform**:
   * Choose "Tomcat" as the platform.
   * Select the version of Java you will be using (Java 8 or Java 11).
3. **Configure Environment**:
   * Use the default sample application for initial setup.
   * Choose custom configuration for advanced settings.
4. **Select IAM Roles**:
   * Ensure Elastic Beanstalk has the necessary service roles (e.g., vprofile-bean-role).
5. **Key Pair**:
   * Create and select an EC2 key pair to enable SSH access for troubleshooting.
6. **VPC Settings**:
   * Choose the default VPC.
   * Ensure instances receive public IPs if needed.
7. **Database Configuration**:
   * For production, manage the database separately using RDS.
8. **Instance Configuration**:
   * Minimum and maximum instance settings for auto-scaling (e.g., 2 instances).
   * Choose instance types (e.g., t3.micro).
9. **Load Balancer Configuration**:
   * Set to public for internet accessibility.
   * Use an application load balancer.
10. **Advanced Settings**:
    * Configure health checks, environment properties, and rolling updates.
    * Rolling updates ensure zero downtime during deployments.

#### Step 3: Verify Elastic Beanstalk Environment

1. **Health Check**:
   * Ensure the environment is in a healthy state.
   * Verify by clicking on the provided URL to see the default application.
2. **EC2 Instances**:
   * Check the EC2 console for the running instances.
   * Note the security groups and auto-scaling group created by Elastic Beanstalk.
3. **Load Balancer and Target Group**:
   * Verify the load balancer and target group configurations.

### Next Steps

In the next video, we will set up an RDS instance to provide a backend database for our application. This involves creating the RDS instance separately and linking it to our Elastic Beanstalk environment to ensure seamless database management and application deployment.

Stay tuned and join me in the next lecture to set up the RDS!

### Summary of Lecture: Deploying a Beanstalk Application with RDS Integration

This lecture outlines the steps to deploy an application on AWS Elastic Beanstalk and connect it to an RDS database. The main goal is to ensure that the application can connect to the RDS instance and that the instance health checks are configured correctly.

#### Steps to Deploy and Configure the Application:

1. **Create RDS Instance:**
   * Select MySQL engine version 5.7.
   * Choose the free tier to ensure cost control.
   * Set instance identifier (vprofile-rds-prod), username (admin), and auto-generate the password.
   * Ensure the instance type is db.t3.micro and storage is 20 GB.
   * Do not connect to an EC2 compute resource during creation.
   * Set up a new security group (vprofile-rds-prod-sg).
   * Configure port 3306 for MySQL.
2. **Configure Security Groups:**
   * Allow port 3306 in the RDS security group from the Beanstalk security group.
   * Obtain the security group ID of the Beanstalk instance.
   * Edit the RDS security group to add an inbound rule for port 3306 from the Beanstalk security group ID.
3. **Initialize the Database:**
   * SSH into the Beanstalk instance.
   * Install the MySQL client using yum install mariadb.
   * Connect to the RDS instance using the MySQL client and verify the connection.
   * Clone the repository containing the SQL file.
   * Deploy the SQL file to the RDS instance using the MySQL client.
4. **Update Health Check Configuration:**
   * In the Beanstalk environment, update the target group health check path to /login.
   * Enable session stickiness in the load balancer settings to ensure consistent routing of user sessions.
5. **Deploy the Application:**
   * Build the application artifact and deploy it to the Beanstalk environment.
   * Apply the changes and monitor the environment health.

#### Key Points:

* Ensure all configurations are set to free tier or minimal resource usage to avoid unnecessary costs.
* Manually configure security groups to understand network connectivity and firewall rules.
* Validate the connection between Beanstalk and RDS by performing database initialization from the Beanstalk instance.
* Adjust health check paths specific to the application to ensure correct health monitoring.
* Enable session stickiness to maintain user session consistency across requests.

### Commands and Configuration Details:

1. **RDS Instance Creation:**

bash

Copy code

# Create an RDS instance with the specified settings.

1. **Security Group Configuration:**

bash

Copy code

# Add inbound rule to RDS security group.

Inbound rule:

- Type: MySQL/Aurora

- Protocol: TCP

- Port: 3306

- Source: Beanstalk security group ID

1. **Initialize Database from Beanstalk Instance:**

bash

Copy code

# SSH into Beanstalk instance

ssh -i path\_to\_key\_pair ec2-user@beanstalk\_instance\_public\_ip

# Install MySQL client

sudo yum install mariadb -y

# Connect to RDS instance

mysql -h rds\_endpoint -u admin -p -D accounts

# Clone the repository and deploy SQL file

git clone repository\_url

cd vprofile-project

mysql -h rds\_endpoint -u admin -p accounts < src/main/resources/db-backup.sql

1. **Update Health Check and Deploy Application:**
   * Change health check path to /login.
   * Enable session stickiness.
   * Apply changes and deploy the application artifact.

By following these steps, you ensure that your application on Beanstalk is correctly configured to connect to an RDS database and passes health checks, leading to a successful deployment.

# Code commit

You’ve done an excellent job transitioning from GitHub to AWS CodeCommit. Now, let's set up the CI/CD pipeline to automate the build and deployment process. We will use AWS CodePipeline, CodeBuild, and CodeDeploy for this purpose. Here’s a step-by-step guide to set up your CI/CD pipeline:

**Step 1: Create a CodeBuild Project**

1. **Navigate to CodeBuild in the AWS Management Console:**
   * Go to the CodeBuild service.
2. **Create a new build project:**
   * Click on "Create build project."
   * **Project configuration:**
     + **Project name:** vprofile-build
     + **Description:** (Optional)
     + **Source provider:** AWS CodeCommit
     + **Repository:** vprofile
     + **Branch:** master (or your preferred branch)
3. **Environment configuration:**
   * **Environment image:** Managed image
   * **Operating system:** Amazon Linux 2
   * **Runtime(s):** Standard
   * **Image:** aws/codebuild/standard:4.0
   * **Service role:** Create a new service role
     + **Role name:** codebuild-vprofile-service-role
4. **Buildspec:**
   * **Build specifications:** Use a buildspec file
     + Ensure there is a buildspec.yml file in your repository's root directory.

yaml

Copy code

version: 0.2

phases:

install:

runtime-versions:

java: corretto11

commands:

- echo Installing dependencies...

pre\_build:

commands:

- echo Logging in to Amazon ECR...

build:

commands:

- echo Build started on `date`

- mvn clean install

post\_build:

commands:

- echo Build completed on `date`

artifacts:

files:

- target/\*.jar

discard-paths: yes

1. **Logs:**
   * Enable CloudWatch logs.
   * Log group: /aws/codebuild/vprofile-build
2. **Create the build project:**
   * Click on "Create build project."

**Step 2: Create an Application in CodeDeploy**

1. **Navigate to CodeDeploy in the AWS Management Console:**
   * Go to the CodeDeploy service.
2. **Create an application:**
   * **Application name:** vprofile-app
   * **Compute platform:** EC2/On-premises
3. **Create a deployment group:**
   * **Deployment group name:** vprofile-deployment-group
   * **Service role:** Create a new service role with the necessary permissions for CodeDeploy.
   * **Deployment type:** In-place
   * **Environment configuration:** Select your EC2 instances (e.g., use tags for identification).
   * **Deployment settings:** Use default settings or customize as per your needs.

**Step 3: Create a Pipeline in CodePipeline**

1. **Navigate to CodePipeline in the AWS Management Console:**
   * Go to the CodePipeline service.
2. **Create a new pipeline:**
   * **Pipeline name:** vprofile-pipeline
   * **Service role:** Create a new service role (default settings are fine).
   * **Advanced settings:** Choose the default options.
3. **Add source stage:**
   * **Source provider:** AWS CodeCommit
   * **Repository name:** vprofile
   * **Branch name:** master (or your preferred branch)
4. **Add build stage:**
   * **Build provider:** AWS CodeBuild
   * **Region:** (your AWS region)
   * **Project name:** vprofile-build
5. **Add deploy stage:**
   * **Deploy provider:** AWS CodeDeploy
   * **Application name:** vprofile-app
   * **Deployment group:** vprofile-deployment-group
6. **Review and create the pipeline:**
   * Click on "Create pipeline."

**Step 4: Update CodeDeploy AppSpec File**

Ensure you have an appspec.yml file in your repository to define the deployment steps.

yaml

Copy code

version: 0.0

os: linux

files:

- source: target/vprofile.war

destination: /usr/local/tomcat/webapps/

hooks:

AfterInstall:

- location: scripts/restart\_server.sh

timeout: 300

runas: root

**Step 5: Final Adjustments and Testing**

1. **Test the Pipeline:**
   * Make a change in your codebase and commit it to trigger the pipeline.
   * Monitor the pipeline’s progress in the CodePipeline console.
2. **Verify Deployment:**
   * Check if the application has been deployed to your EC2 instances.
   * Ensure the application is running correctly.

By following these steps, you should have a fully functional CI/CD pipeline using AWS CodeCommit, CodeBuild, CodeDeploy, and CodePipeline.

# CodeBuild

### Step-by-Step Instructions to Set Up CodeBuild and Deploy to AWS Elastic Beanstalk

#### 1. Create a CodeBuild Project

1. **Navigate to CodeBuild:**
   * In your AWS Management Console, go to the CodeBuild service.
2. **Create a New Build Project:**
   * Click on "Create build project."
   * **Project Name:** vprofile-build
   * **Description:** (Optional)
   * **Source Provider:** CodeCommit
   * **Repository:** vprofile
   * **Branch:** vp-rem
3. **Environment Configuration:**
   * **Environment Image:** Managed image
   * **Operating System:** Ubuntu
   * **Runtime(s):** Standard
   * **Image:** aws/codebuild/standard:4.0
   * **Service Role:** Create a new service role
     + **Role Name:** codebuild-vprofile-service-role
4. **Buildspec File:**
   * Ensure you have a buildspec.yml file in your repository. Here is the example buildspec.yml tailored for your project:

yaml

Copy code

version: 0.2

env:

variables:

DB\_USERNAME: "admin"

DB\_PASSWORD: "yourpassword"

RDS\_ENDPOINT: "your-rds-endpoint"

phases:

install:

runtime-versions:

java: corretto11

commands:

- echo Installing dependencies...

- apt-get update -y

- apt-get install -y jq

pre\_build:

commands:

- echo Pre-build started on `date`

- sed -i 's/JDBC\_PASSWORD=.\*/JDBC\_PASSWORD=${DB\_PASSWORD}/' src/main/resources/application.properties

- sed -i 's/JDBC\_USERNAME=.\*/JDBC\_USERNAME=${DB\_USERNAME}/' src/main/resources/application.properties

- sed -i 's/DB\_URL=.\*/DB\_URL=jdbc:mysql://${RDS\_ENDPOINT}:3306/vprofile/' src/main/resources/application.properties

build:

commands:

- echo Build started on `date`

- mvn clean install

post\_build:

commands:

- echo Build completed on `date`

artifacts:

files:

- target/\*.jar

- appspec.yml

- scripts/\*\*/\*

discard-paths: yes

1. **Artifacts:**
   * **Artifacts Type:** Amazon S3
   * **Bucket Name:** Select your S3 bucket (ensure it’s in the same region)
   * **Artifact Packaging:** None
2. **Logs:**
   * **CloudWatch Logs:** Enable
   * **Log Group Name:** /aws/codebuild/vprofile-build
   * **Log Stream Name:** build-logs
3. **Create the Build Project:**
   * Click on "Create build project."

#### 2. Update the AppSpec File for CodeDeploy

Ensure your repository has an appspec.yml file to define the deployment steps. Here’s an example:

yaml

Copy code

version: 0.0

os: linux

files:

- source: target/vprofile.war

destination: /usr/local/tomcat/webapps/

hooks:

AfterInstall:

- location: scripts/restart\_server.sh

timeout: 300

runas: root

Ensure you have a scripts/restart\_server.sh file to restart your application server after deployment:

bash

Copy code

#!/bin/bash

sudo systemctl restart tomcat

#### 3. Create an Application in CodeDeploy

1. **Navigate to CodeDeploy:**
   * Go to the CodeDeploy service.
2. **Create an Application:**
   * **Application Name:** vprofile-app
   * **Compute Platform:** EC2/On-premises
3. **Create a Deployment Group:**
   * **Deployment Group Name:** vprofile-deployment-group
   * **Service Role:** Create a new service role with necessary permissions.
   * **Deployment Type:** In-place
   * **Environment Configuration:** Select your EC2 instances (e.g., using tags).
   * **Deployment Settings:** Use default settings or customize as needed.

#### 4. Create a Pipeline in CodePipeline

1. **Navigate to CodePipeline:**
   * Go to the CodePipeline service.
2. **Create a New Pipeline:**
   * **Pipeline Name:** vprofile-pipeline
   * **Service Role:** Create a new service role (default settings are fine).
3. **Add Source Stage:**
   * **Source Provider:** AWS CodeCommit
   * **Repository Name:** vprofile
   * **Branch Name:** vp-rem
4. **Add Build Stage:**
   * **Build Provider:** AWS CodeBuild
   * **Region:** Your AWS region
   * **Project Name:** vprofile-build
5. **Add Deploy Stage:**
   * **Deploy Provider:** AWS CodeDeploy
   * **Application Name:** vprofile-app
   * **Deployment Group:** vprofile-deployment-group
6. **Review and Create the Pipeline:**
   * Click on "Create pipeline."

#### 5. Final Adjustments and Testing

1. **Test the Pipeline:**
   * Make a change in your codebase and commit it to trigger the pipeline.
   * Monitor the pipeline’s progress in the CodePipeline console.
2. **Verify Deployment:**
   * Check if the application has been deployed to your EC2 instances.
   * Ensure the application is running correctly.

### Summary

By following these steps, you will set up a CI/CD pipeline using AWS CodeCommit, CodeBuild, CodeDeploy, and CodePipeline. This pipeline will fetch your code from CodeCommit, build it using CodeBuild, and deploy it to your EC2 instances using CodeDeploy.

Fantastic! You have successfully tested your build job and integrated all the elements of your CI/CD pipeline using AWS services. Let's summarize the steps for future reference:

### Summary of Steps

#### 1. Create and Test CodeBuild Project

1. **Create a CodeBuild Project:**
   * **Project Name:** vprofile-build
   * **Source Provider:** CodeCommit
   * **Repository:** vprofile
   * **Branch:** vp-rem
   * **Environment Configuration:**
     + **Image:** aws/codebuild/standard:4.0
     + **Operating System:** Ubuntu
   * **Buildspec File:**

yaml

Copy code

version: 0.2

env:

variables:

DB\_USERNAME: "admin"

DB\_PASSWORD: "yourpassword"

RDS\_ENDPOINT: "your-rds-endpoint"

phases:

install:

runtime-versions:

java: corretto11

commands:

- echo Installing dependencies...

- apt-get update -y

- apt-get install -y jq

pre\_build:

commands:

- echo Pre-build started on `date`

- sed -i 's/JDBC\_PASSWORD=.\*/JDBC\_PASSWORD=${DB\_PASSWORD}/' src/main/resources/application.properties

- sed -i 's/JDBC\_USERNAME=.\*/JDBC\_USERNAME=${DB\_USERNAME}/' src/main/resources/application.properties

- sed -i 's/DB\_URL=.\*/DB\_URL=jdbc:mysql://${RDS\_ENDPOINT}:3306/vprofile/' src/main/resources/application.properties

build:

commands:

- echo Build started on `date`

- mvn clean install

post\_build:

commands:

- echo Build completed on `date`

artifacts:

files:

- target/\*.jar

- appspec.yml

- scripts/\*\*/\*

discard-paths: yes

* + **Artifacts:** Upload to S3 bucket
  + **Logs:** Enable CloudWatch Logs

1. **Test the CodeBuild Project:**
   * Click on "Start Build."
   * Monitor the build process in CloudWatch Logs to ensure it completes successfully.

#### 2. Create CodeDeploy Application and Deployment Group

1. **Create a CodeDeploy Application:**
   * **Application Name:** vprofile-app
   * **Compute Platform:** EC2/On-premises
2. **Create a Deployment Group:**
   * **Deployment Group Name:** vprofile-deployment-group
   * **Service Role:** Create with necessary permissions
   * **Environment Configuration:** Select EC2 instances

#### 3. Create CodePipeline

1. **Create a New Pipeline:**
   * **Pipeline Name:** vprofile-pipeline
   * **Service Role:** Create a new service role
2. **Add Source Stage:**
   * **Source Provider:** CodeCommit
   * **Repository Name:** vprofile
   * **Branch Name:** vp-rem
3. **Add Build Stage:**
   * **Build Provider:** CodeBuild
   * **Project Name:** vprofile-build
4. **Add Deploy Stage:**
   * **Deploy Provider:** CodeDeploy
   * **Application Name:** vprofile-app
   * **Deployment Group:** vprofile-deployment-group
5. **Create the Pipeline:**
   * Review and create the pipeline.
   * Monitor the pipeline execution and check the deployment status in Elastic Beanstalk.

#### 4. Verify and Test the CI/CD Pipeline

1. **Trigger a Build and Deploy:**
   * Make a code change and commit it to the vp-rem branch.
   * Monitor the pipeline to ensure it triggers the build and deploy stages.
2. **Verify Deployment:**
   * Check the Elastic Beanstalk environment to ensure the new version is deployed.
   * Verify the application is running correctly.

### Clean Up Resources

1. **Delete Elastic Beanstalk Environment:**
   * Remove the link from the RDS security group.
   * Delete the Beanstalk environment.
2. **Delete RDS Instance:**
   * Ensure no connections are active.
   * Delete the RDS instance to avoid unnecessary charges.
3. **Retain CodeCommit, CodeBuild, and CodePipeline:**
   * These services incur minimal charges when not in use.

By following these steps, you have set up a robust CI/CD pipeline using AWS services. This pipeline will help automate the process of building, testing, and deploying your application, making your development workflow more efficient and reliable.