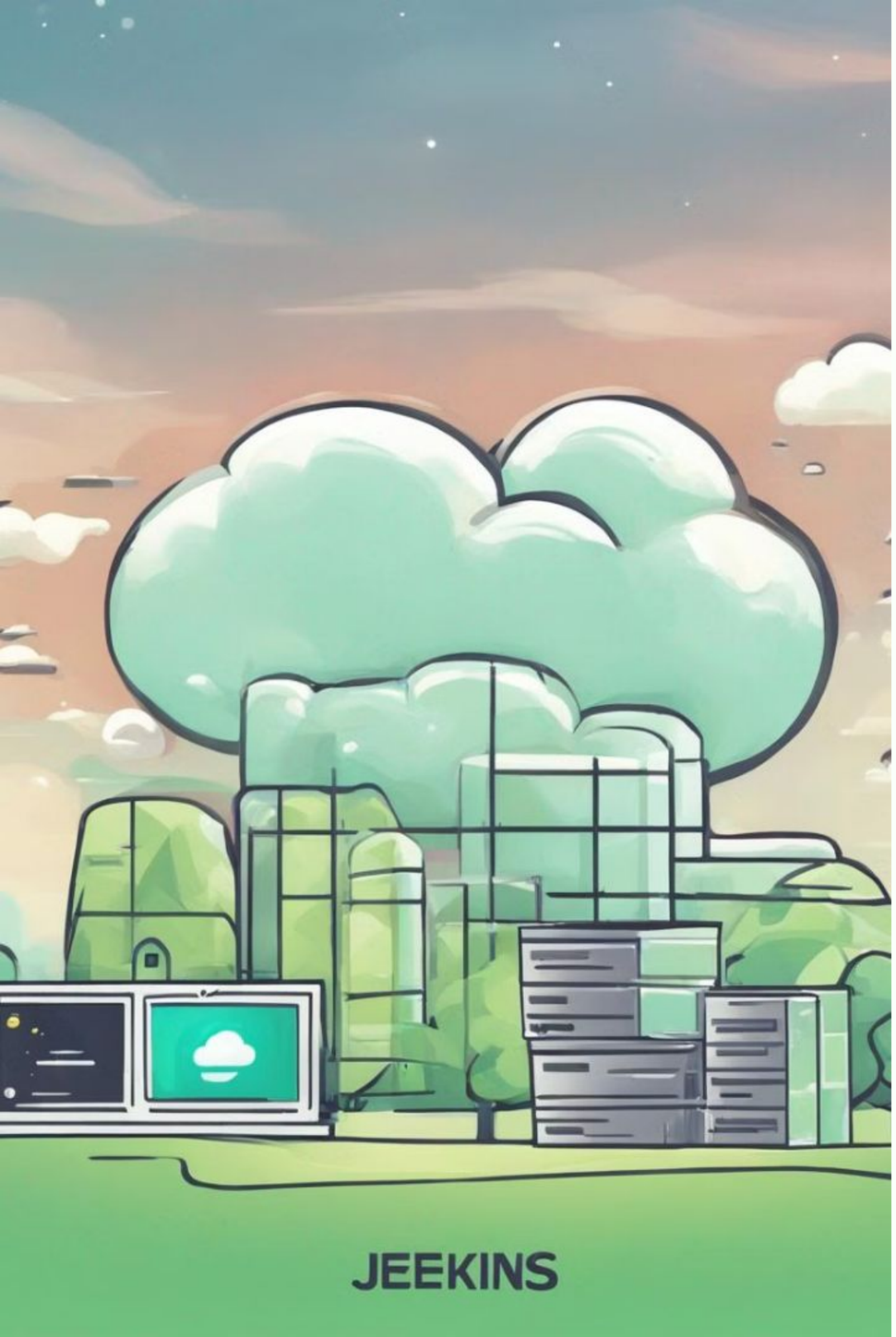


Deploying React Apps to AWS with Jenkins

This presentation will guide you through the process of deploying your React application to AWS using Jenkins. We'll cover setting up a Jenkins server, configuring it for deployment, and automating the entire process.

by Abhijith H

Setting up a Jenkins Server

1

Install Java

Jenkins requires Java to run. Download and install the latest LTS version of Java from Oracle's website.

2

Download Jenkins

Download the latest Jenkins WAR file from the official Jenkins website.

3

Run Jenkins

Run the WAR file using the command `java -jar jenkins.war` in your terminal.

4

Configure Plugins

Install necessary plugins like the AWS plugin for interacting with AWS services.

Configuring Jenkins for React App Deployment

Create a New Project

In Jenkins, create a new project to manage your React app deployment.

Configure Git Integration

Connect your project's Git repository to Jenkins to fetch code updates for deployment.

Define Build Steps

Specify the build commands, such as `npm install` and `npm run build`, to prepare your React app for deployment.

Configure Deployment Targets

Specify the AWS resources, like an S3 bucket or EC2 instance, where your React app will be deployed.

Automating the Build Process with Jenkins

1

Code Commit

When you push code changes to your Git repository, Jenkins triggers the build process.

2

Code Checkout

Jenkins fetches the latest code from your Git repository.

3

Build and Test

Jenkins executes the build commands and runs tests to ensure code quality.

4

Deployment

If the build and tests pass, Jenkins deploys the built React app to AWS.

Containerizing the React App with Docker

Create Dockerfile

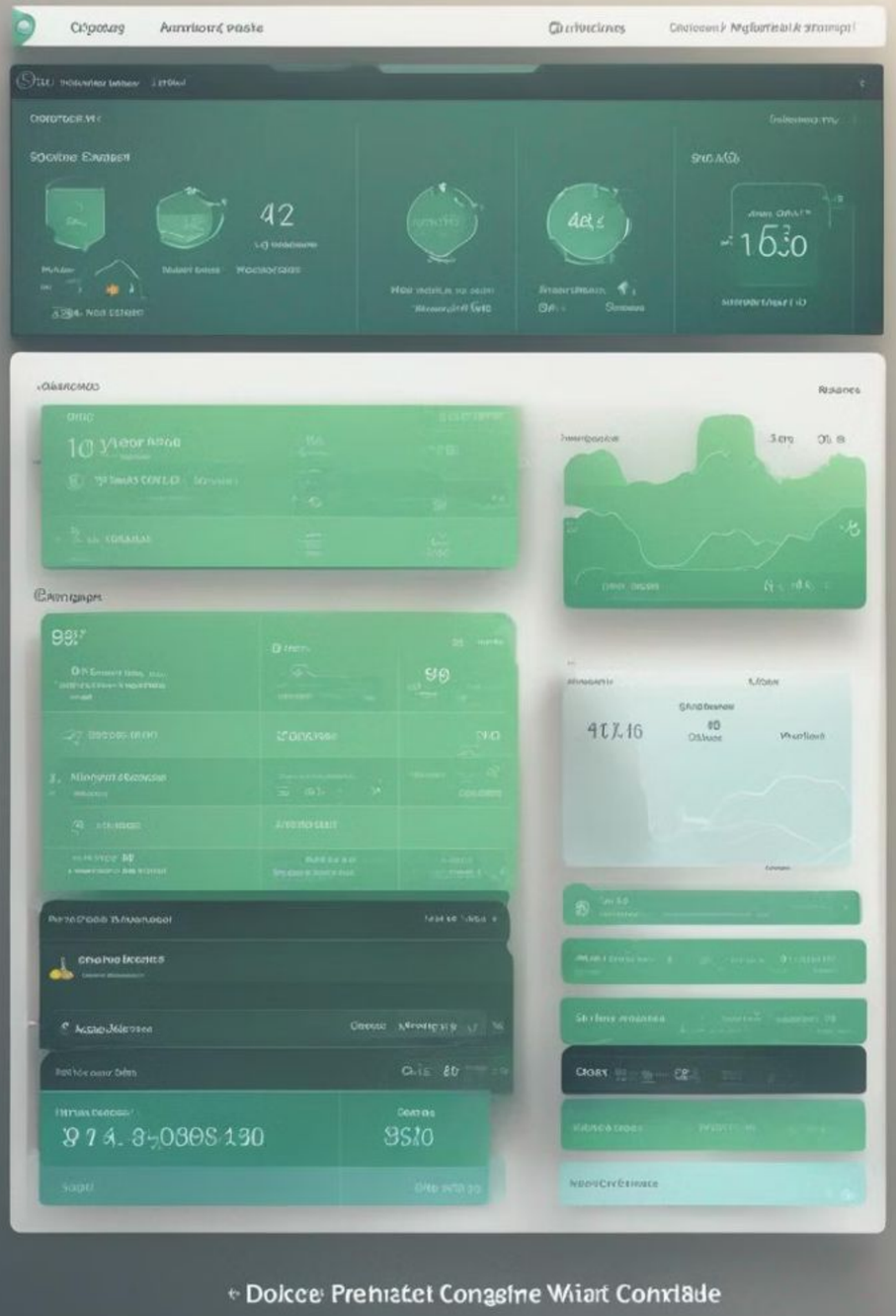
Write a Dockerfile that defines the container's environment, including the base image, dependencies, and build commands.

Build Docker Image

Use the command `docker build -t [image-name]:[tag]` to build a Docker image from the Dockerfile.

Run Docker Container

Run the Docker container using the command `docker run -p [port]:[port] [image-name]:[tag]` to start your React app within the container.



Deploying the Docker Container to AWS

1

AWS Elastic Container Service (ECS)

Deploy your containerized React app using AWS ECS, a managed container orchestration service.

2

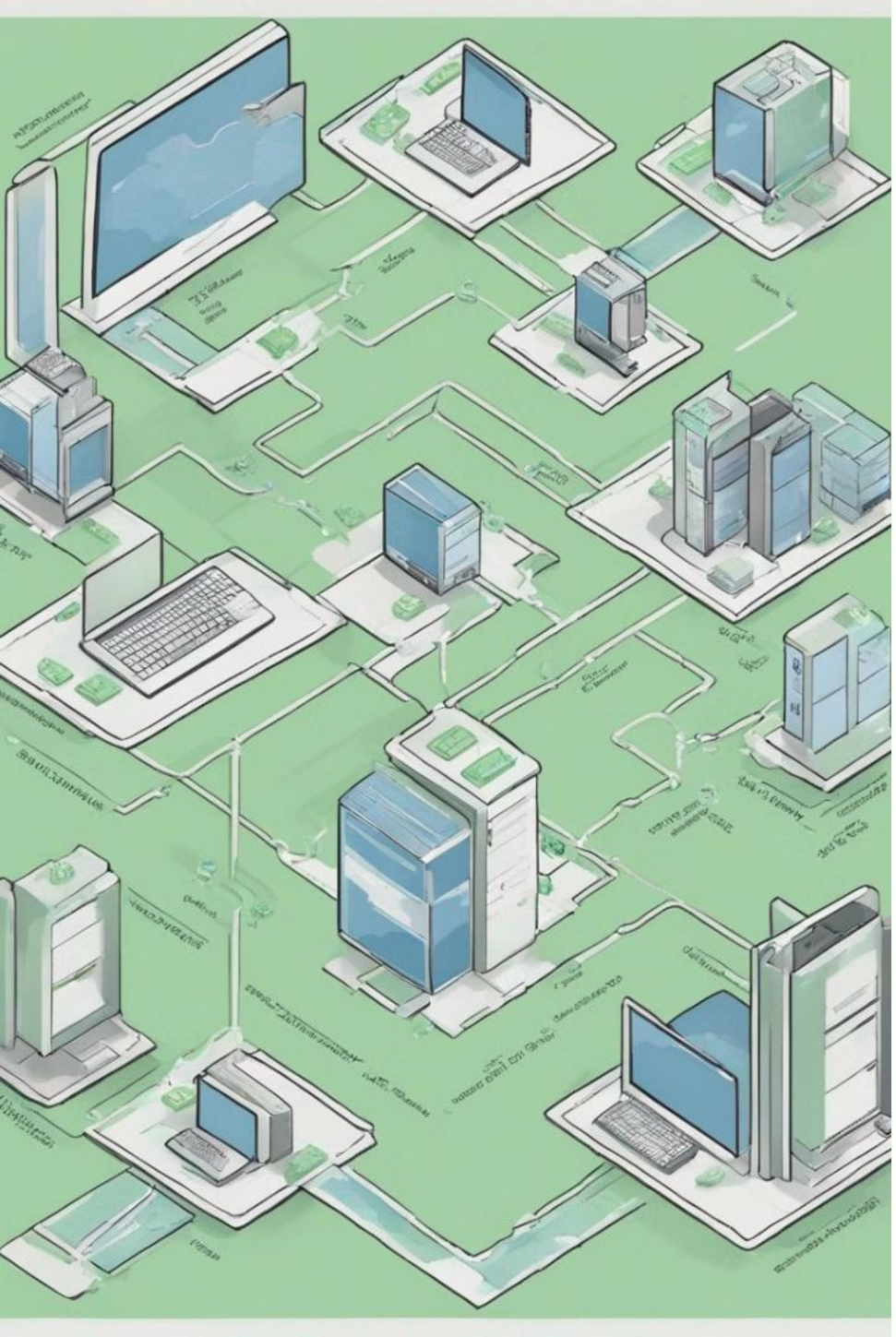
AWS Fargate

Use AWS Fargate to run your containers without managing the underlying infrastructure.

3

AWS Elastic Beanstalk

Utilize AWS Elastic Beanstalk to simplify deployment by handling infrastructure and configuration.



Configuring AWS Services for React App Hosting

Service	Description
Amazon S3	Store static assets, such as HTML, CSS, and JavaScript files, for your React app.
Amazon CloudFront	Deliver content to users globally with high availability and performance.
Amazon Route 53	Manage DNS records and route traffic to your deployed React app.
Amazon EC2	Provision virtual servers to run your React app if dynamic content or backend functionality is required.

Integrating Jenkins and AWS for Continuous Deployment

AWS Credentials

Configure Jenkins with AWS credentials to access and interact with AWS services.

AWS Plugin

Install the AWS plugin in Jenkins to automate interactions with AWS services.

Jenkins Pipelines

Define Jenkins pipelines to orchestrate the deployment process, including build, test, and deployment steps.

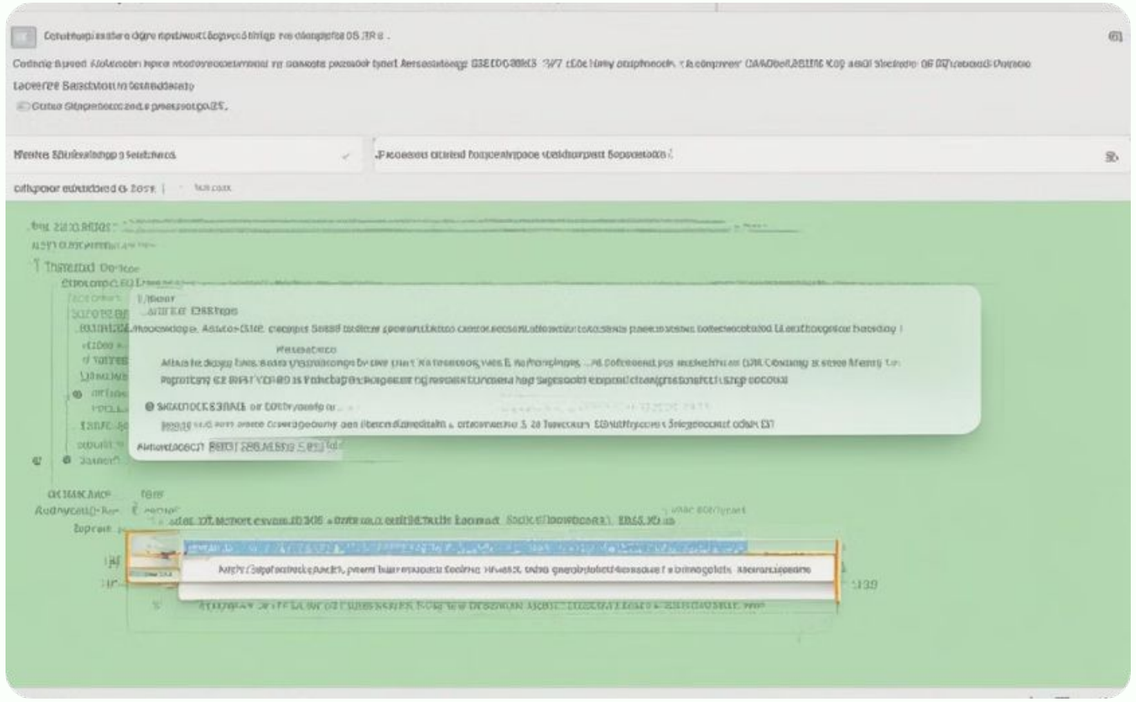
AWS Deployment Targets

Specify the AWS resources, such as S3 buckets or EC2 instances, as deployment targets in your pipelines.

Monitoring and Troubleshooting the Deployed React App

Amazon CloudWatch

Monitor your deployed React app's performance and health using AWS CloudWatch.



Amazon CloudTrail

Use AWS CloudTrail to track API calls and events related to your deployed React app.

Conclusion and Best Practices

Deploying React apps to AWS using Jenkins is a powerful and efficient way to achieve continuous deployment. Utilize best practices like code modularity, automated testing, and comprehensive monitoring to ensure a seamless and successful deployment process.