**AWS-Based DevOps Plan for Web Projects**

**1️. Infrastructure Setup (Server & Packages Management)**

**AWS Services Used:**

✅ **EC2 Instances** (For hosting the application)  
✅ **RDS / Aurora** (For database management)  
✅ **S3** (For file storage)  
✅ **Route 53** (For DNS management)  
✅ **CloudFront** (For CDN & caching)  
✅ **IAM Roles & Policies** (For security & access control)  
✅ **Systems Manager (SSM)** (For managing packages & configurations)

**Implementation Steps:**

1. **Provision AWS EC2**
   * Choose an **Amazon Linux 2** or **Ubuntu 22.04** instance.
   * Install required dependencies like **Nginx, PHP, MySQL, Redis, Node.js, etc.**
   * Use **Amazon Machine Image (AMI)** for easy replication.
2. **Database Setup:**
   * Use **Amazon RDS (MySQL/PostgreSQL)** or **Aurora**.
   * Enable **Automated Backups** & **Multi-AZ Deployments** for reliability.
3. **Auto-Scaling & Load Balancing:**
   * **Application Load Balancer (ALB)** for better traffic distribution.
   * **Auto Scaling Groups (ASG)** to dynamically adjust capacity.
4. **File Storage & Caching:**
   * Store **static assets** in **Amazon S3** and serve via **CloudFront**.
   * Enable **Redis/Memcached** for caching.
5. **Security & Access Control:**
   * Create **IAM roles** for EC2 instances.
   * Use **AWS Systems Manager (SSM)** for **package management** & **SSH-less access**.

**2️. GitHub Repository Branching Strategy**

**Branch Structure:**

- main (Stable Production Code)

- dev (Development & Staging)

- AWS (Dedicated AWS environment)

- feature/\* (Feature branches)

✅ **AWS branch** is used for development and deployments related to AWS.  
✅ **Pull Requests (PRs)** must be reviewed before merging into main.  
✅ Protect main & AWS branches from direct push, enforce **PR approvals**.

**GitHub Actions for Code Validation**

* Setup **pre-commit hooks** for linting & formatting.
* Run **GitHub Actions** to test the code before merging.

**3️. CI/CD Automation with Jenkins**

✅ **Jenkins (on AWS EC2)** – Install via **Docker** or direct setup.  
✅ **GitHub Webhooks** – Trigger pipelines on push.  
✅ **Terraform or AWS CLI** – Infrastructure as Code (IaC).  
✅ **Docker & Kubernetes (Optional)** – Containerized deployments.

1. **Jenkins Pipeline Configuration:**
   * **Fetch AWS branch** from GitHub.
   * **Run unit tests, linting, and security scans** (PHPStan, ESLint, SonarQube).
   * **Build the application** (Laravel, Vue.js, React, etc.).
   * **Push artifacts to AWS S3 or ECR**.
   * **Deploy to EC2 using SSH or SSM**.
   * **Automate database migrations** (php artisan migrate).
   * **Restart services using systemctl or Docker**.
2. **Deployment Strategies:**
   * **Blue/Green Deployment** using two EC2 instances.
   * **Rolling Updates** to minimize downtime.
   * **Canary Deployments** to test new features before full rollout.
3. **Rollback Strategy:**
   * Use **Amazon S3** & **Versioning** to store previous builds.
   * Enable **database snapshots** before deploying changes.
   * Implement **Jenkins Job for Rollback**.

**4️. Monitoring & Logging with Datadog**

**Why Datadog?**

✅ **Real-time Performance Monitoring**  
✅ **Centralized Log Management**  
✅ **Error Tracking & Alerting**  
✅ **Application Tracing**

**Integration Steps:**

1. **Install Datadog Agent** on EC2:

DD\_API\_KEY=<YOUR\_API\_KEY> bash -c "$(curl -L https://s3.amazonaws.com/dd-agent/scripts/install\_script.sh)"

1. **Enable Log Collection**
   * Monitor logs from **Nginx, Laravel, MySQL, Redis, Docker**.
   * Configure **custom dashboards** for CPU, Memory, Disk, and API response times.
2. **Set Up Alerts:**
   * Alert on **high CPU usage, slow database queries, failed deployments**.
   * Use **Slack, Email, or SMS** for notifications.

**5️. Security & Compliance**

**AWS Security Best Practices**

✔ Enable **AWS GuardDuty** for threat detection.  
✔ Use **AWS WAF & Shield** for DDoS protection.  
✔ Store **Secrets in AWS Secrets Manager** instead of .env files.  
✔ Enforce **multi-factor authentication (MFA)** for GitHub & AWS accounts.  
✔ Implement **automated security scans** in Jenkins using **SonarQube**.

**6️. Cost Optimization**

**Cost-Effective AWS Usage**

✔ **Use AWS Free Tier** where possible.  
✔ **Choose AWS EC2 Spot Instances** for cost savings.  
✔ **Enable Auto-Scaling** to scale down during low traffic hours.  
✔ **Use AWS Compute Savings Plans** for predictable workloads.  
✔ **Optimize Datadog Logging** to avoid excessive log storage costs.

**Final Tech Stack Summary**

| **Component** | **Tool / Service** |
| --- | --- |
| **Infrastructure** | AWS EC2, RDS, S3, Route 53, CloudFront |
| **Version Control** | GitHub (AWS branch) |
| **CI/CD** | Jenkins (EC2) |
| **Containerization (Optional)** | Docker, Kubernetes |
| **Monitoring & Logging** | Datadog |
| **Security** | AWS IAM, GuardDuty, WAF, Secrets Manager |
| **Cost Optimization** | Spot Instances, Compute Savings Plans |

* **Key Benefits of This Plan**

✔ **Automated deployments & reduced manual work**  
✔ **Better collaboration using GitHub branching strategy**  
✔ **Quicker time to market with CI/CD pipelines**  
✔ **Real-time monitoring & alerting for proactive issue resolution**  
✔ **Cost-effective AWS usage to minimize expenses**  
✔ **Scalability & high availability for improved user experience**

**1️. Terraform: AWS Infrastructure as Code (IaC)**

**Terraform Setup**

* **Creates an EC2 instance for hosting the web app**
* **Configures an RDS database (MySQL)**
* **Sets up an S3 bucket for static files**
* **Configures security groups for firewall rules**
* **Stores secrets in AWS Secrets Manager**

**Apply Terraform**

terraform init

terraform apply -auto-approve

**2️. Jenkinsfile: CI/CD Pipeline**

**Features:**

✅ Fetches code from **AWS branch** in GitHub  
✅ Runs **unit tests, linting, security scans**  
✅ Builds the application & pushes artifacts to **S3**  
✅ Deploys to **EC2 via SSH or AWS Systems Manager**  
✅ **Uses AWS Secrets Manager** for sensitive data  
✅ Sends **Slack notifications** for success/failure

**Jenkinsfile**

pipeline {

agent any

environment {

AWS\_REGION = 'us-east-1'

S3\_BUCKET = 'my-webapp-static-assets'

EC2\_INSTANCE\_ID = 'i-1234567890abcdef'

DB\_CREDENTIALS = credentials('aws-secretsmanager-db') // Use Jenkins stored credentials

SLACK\_CHANNEL = '#deployments'

SLACK\_CREDENTIALS = credentials('slack-webhook-url') // Slack webhook stored in Jenkins

}

stages {

stage('Checkout Code') {

steps {

git branch: 'AWS', url: 'https://github.com/myorg/myrepo.git'

}

}

stage('Run Tests') {

steps {

sh 'composer install'

sh 'php artisan test'

}

}

stage('Security Scan') {

steps {

sh 'composer require --dev friendsofphp/php-cs-fixer'

sh 'vendor/bin/php-cs-fixer fix --dry-run'

}

}

stage('Build & Package') {

steps {

sh 'zip -r app.zip .'

}

}

stage('Upload to S3') {

steps {

sh "aws s3 cp app.zip s3://$S3\_BUCKET/app.zip --region $AWS\_REGION"

}

}

stage('Deploy to EC2') {

steps {

sshagent(['aws-ec2-ssh']) {

sh """

ssh -o StrictHostKeyChecking=no ec2-user@${EC2\_INSTANCE\_ID} << 'EOF'

sudo systemctl stop nginx

aws s3 cp s3://$S3\_BUCKET/app.zip /var/www/html/app.zip

cd /var/www/html

unzip -o app.zip

sudo systemctl restart nginx

EOF

"""

}

}

}

stage('Post-Deployment Verification') {

steps {

sh "curl -f http://myapp.com || exit 1"

}

}

stage('Notify Slack') {

steps {

sh """

curl -X POST --data-urlencode "payload={\\"channel\\": \\"$SLACK\_CHANNEL\\", \\"text\\": \\"Deployment completed successfully!\\"}" $SLACK\_CREDENTIALS

"""

}

}

}

post {

failure {

sh """

curl -X POST --data-urlencode "payload={\\"channel\\": \\"$SLACK\_CHANNEL\\", \\"text\\": \\"Deployment failed! Check logs.\\"}" $SLACK\_CREDENTIALS

"""

}

}

}

**3️. Storing Secrets in AWS Secrets Manager & Jenkins**

**AWS Secrets Manager (DB Credentials & API Keys)**

1. Store credentials:

sh

CopyEdit

aws secretsmanager create-secret --name aws-secretsmanager-db --secret-string '{"db\_username":"admin","db\_password":"my\_secure\_password"}'

1. Retrieve credentials inside Jenkins:

sh

CopyEdit

aws secretsmanager get-secret-value --secret-id aws-secretsmanager-db

**Jenkins Secrets Setup**

1. **AWS IAM Credentials** → Store under **Manage Jenkins → Credentials**
   * ID: aws-credentials
   * Username: AWS\_ACCESS\_KEY\_ID
   * Password: AWS\_SECRET\_ACCESS\_KEY
2. **SSH Key for EC2 Deployment** → Store under **Manage Jenkins → Credentials**
   * ID: aws-ec2-ssh
   * Private Key: **(Paste your .pem key here)**
3. **Slack Webhook** → Store under **Manage Jenkins → Credentials**
   * ID: slack-webhook-url
   * Secret: **Paste Slack Webhook URL**