AWS-Based DevOps Plan for Web Projects

Goals:

✅ Better Collaboration
✅ Quicker Time to Market
✅ Reduced Manual Errors
✅ Optimal Quality Software Delivery
✅ Improved Reliability
✅ Enhanced User Experience
✅ More Stable & Resilient Apps

---

1. 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**.

---

2. 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.

---

3. CI/CD Automation with Jenkins

Setup & Tools:

✅ **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.

CI/CD Workflow:

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**.

---

4. 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)"
```

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

---

5. 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**.

---

6. 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**

Next Steps

✓ **Set up the AWS infrastructure** using Terraform or AWS CLI.
✓ **Configure Jenkins & GitHub Webhooks** for automated deployments.
✓ I**mplement Datadog Monitoring** for logs, metrics, and alerts.
✓ **Test CI/CD Pipeline & Deployment Strategies** before production.