Deployment Pipeline Description for LawVriksh Blog Management System

1. Development Environment

- Developers clone the repository from a version control system (e.g., GitHub).
- They set up the local development environment with a virtual environment, install dependencies (`requirements.txt`), and configure environment variables in a `.env` file.
- Local development uses a development instance of NeonDB or a shared testing database.
- Developers perform coding, manual testing, and use Postman or API clients to validate API functionality.
- Feature branches are created for new development and bug fixes.

2. Version Control and Continuous Integration (CI)

- Code is pushed to remote branches on GitHub.
- Pull Requests (PRs) are created to merge features or fixes into the main branch.
- A CI pipeline (e.g., GitHub Actions, GitLab CI, Jenkins) is triggered on PRs and commits to main.
- CI tasks include:
- Linting and code style checks using tools like `black` and `flake8`.
- Running unit and integration tests using `pytest`.
- Security and static code analysis.
- If all checks pass, PRs are approved and merged into main.

3. Containerization and Build

- The application is containerized using a Dockerfile, which builds a lightweight container image with the FastAPI app, dependencies, and environment setup.
- The CI pipeline builds this Docker image on each commit to main.
- The built image is tagged (e.g., with the commit SHA or version number) and pushed to a container registry such as Docker Hub, Amazon ECR, or Google Container Registry.

4. Staging Environment and Continuous Deployment (CD)

- A Continuous Deployment pipeline deploys the newly built Docker image to a staging environment.
- Staging is a replica of production infrastructure but isolated for safe testing.
- Kubernetes, Docker Compose, or a container service (e.g., AWS ECS, Azure Container Instances) is used to run the container.
- Environment variables such as NeonDB connection URL, JWT secrets, and others are securely injected using secrets management.
- Smoke tests and automated API checks are run against staging for validation.
- Stakeholders perform manual acceptance testing on staging.

5. Production Deployment

- Upon successful validation in staging, the same Docker image is promoted to the production environment, preventing differences between staging and production.
- Production deployment may happen manually via promotion or automatically through CD pipelines with appropriate approvals.
- Load balancing and HTTPS termination components (e.g., NGINX, Traefik, or cloud load balancers) manage incoming traffic securely.
- Monitoring and logging services (e.g., Prometheus, ELK stack, Datadog) are set up for observability.
- Backups for the NeonDB production database are in place with recovery strategies.

6. Rollbacks and Maintenance

- Deployments are versioned; rollback to previous stable versions is possible by redeploying older Docker images.
- Regular security updates on dependencies and infrastructure are applied.
- CI/CD pipeline is maintained for reliability and improved automation.
- Documentation and runbooks for deployment and operational procedures are kept up to date.