📘 Zero Downtime Deployments

For Developers Who Ship Like Pros

Author: Sandip Mhaske
Full Stack .NET + Angular + AWS Developer

🔹 1. The Why

Why do deployments break?

Deployments often introduce temporary or prolonged **downtime** due to:

- Restarted services during upgrades
- Incomplete environment setups
- Failing dependencies during rollout
- In-flight user sessions being disrupted

What is “Zero Downtime”?

Zero Downtime means your users:

- Don’t face broken interfaces or white screens
- Don’t get logged out or face 503 errors
- Continue using the application seamlessly
- Don’t even notice that a deployment happened

Why does it matter?

- ✅ Builds trust with your users
- ✅ Reduces emergency fixes and panic rollbacks
- ✅ Allows smaller, safer, and more frequent deployments
- ✅ Boosts confidence in releasing features faster

🔹 2. Core Strategies

To achieve Zero Downtime, you need deployment strategies that are **gradual, reversible, and observable**.

a. Blue-Green Deployments

- Two identical environments: **Blue (Live)** and **Green (New)**
- You deploy to Green, test it thoroughly, then reroute traffic
- If issues arise, simply switch back to Blue

b. Canary Releases

- Roll out to a **small percentage of users** (1–10%)
- Monitor logs, performance, and user impact
- Gradually increase rollout if all goes well
- Roll back if anomalies or errors spike

c. Rolling Updates

- Deploy updates **in waves** across your infrastructure
- Helps maintain availability in distributed systems
- Each new instance is warmed up before receiving live traffic

d. Feature Toggles

- New code is shipped but hidden behind feature flags
- Toggle features on/off without redeploying
- Useful for gradual enablement, A/B testing, or experimentation

🔹 3. Stack-Specific Techniques

.NET (IIS/Kestrel)

- Use **Application Initialization** to warm up the app pool
- Avoid `app_offline.htm`, which causes sudden app stoppage
- Integrate health checks with load balancers for routing

Angular (Frontend)

- Deploy static assets via **AWS S3** or **CloudFront**
- Enable **cache busting** with versioned builds
- Invalidate CloudFront cache on each deploy to prevent stale assets

AWS-Specific Tools I Recommend

- **AWS CodeDeploy**: Native support for Blue-Green and Canary
- **Elastic Beanstalk / ECS**: For rolling update configurations
- **GitHub Actions**: Automate CI/CD with environment-specific workflows

🔹 4. CI/CD Flow with Rollback Support

Ideal CI/CD Flow

1. Code pushed to GitHub
2. Build + Test pipeline executes
3. Deploy to **Staging**
4. Manual or automated approval
5. Deploy to **Production**
6. Auto-health checks run
7. Observability tools monitor live usage

Rollback Plan Essentials

- Keep the previous production version live (e.g., Blue)
- Rollback triggered by error spikes, failed health checks, or latency
- Use deployment tracking and tagging to instantly revert

🔹 5. Testing, Monitoring, and Observability

a. Smoke Testing

- Light and fast post-deployment validation
- Run tests on critical APIs and UI flows

b. Health Checks

- Load balancers should only send traffic to **healthy** instances
- Implement readiness and liveness probes

c. Rollback Preparedness

- Don't wait for failure to plan rollback
- Keep rollback automation integrated into the CI/CD flow

d. Observability Stack

- **CloudWatch**: Logs, metrics, alarms
- **Sentry**: Frontend error tracking, stack traces
- **New Relic / Datadog**: Application Performance Monitoring

---

🔹 6. My Real-World Lessons

As a full stack developer managing production deployments, I've faced:

- White screens after Angular build deployment
- Users losing sessions mid-form submission
- IIS restarts killing .NET app pools
- Features half-enabled causing partial crashes

These experiences taught me that **code quality means nothing if deployment is flawed**.

Key lessons I follow now:

- Deploy often, but deploy small
- Automate everything but **verify results**
- Don't trust what passed CI — **observe production**
- Keep the rollback option one click away
- Combine **technical discipline** with **user empathy**

🔹 7. Conclusion & Call to Action

Zero Downtime Deployments aren't magic. They're a **process**. A **discipline**. A **commitment** to user experience.

If this guide sparked ideas or helped you level up, I'd love to hear from you.

📬 DM me if:

- You're planning to build a CI/CD workflow
- Need AWS-based Blue-Green deployments
- Want help introducing toggles or monitoring to your stack

🔁 And follow me for more real-world breakdowns — explained with simplicity and precision.

Let's deploy like pros — without breaking anything.

🔖 Document Info

Title: Zero Downtime Deployments – For Developers Who Ship Like Pros
Created By: Sandip Mhaske
Date: July 2025
Target Audience: Developers, DevOps Engineers, Tech Leads, CTOs