

Cloud, Containers, Load Balancing, and Release Engineering

Lesson Guide · Lab / Exam Guide

 LearnWisej

Lesson Guide

IIS, Kestrel, reverse proxies, containers, sticky sessions, health checks, CI/CD, runbooks. What this means for an advanced Wisej.NET team — and how to read this module.



Production focus

IIS, Kestrel, reverse proxies, containers, sticky sessions, health checks, CI/CD, runbooks.

1. The production problem

Advanced Wisej.NET applications must move beyond developer machines into repeatable environments with configuration discipline, health checks, load balancing, rollback, and operational ownership.

2. Key concepts

Concept	Why it matters
IIS	The Windows web server that hosts Wisej.NET applications in many enterprises, with application pools, recycling rules, and websocket support to configure.
Kestrel	The cross-platform ASP.NET Core web server that runs a Wisej.NET application directly, usually behind a reverse proxy in production.
NGINX/Apache reverse proxy	A front-end server that terminates TLS, forwards websocket traffic, and routes requests to one or more application instances.
Azure/AWS	Cloud platforms that host the application as app services, containers, or virtual machines, each with its own scaling and configuration model.
Docker	Packages the application and its runtime into one image, so the same build runs identically on a developer machine, a test server, and the cloud.
HealthCheck.json	The Wisej.NET health-check configuration that lets load balancers and orchestrators probe whether an instance is ready to serve sessions.
sticky sessions	Load-balancer affinity that routes every request from one user to the same server instance, which is required because session state lives in server memory.
CI/CD	Automated build, test, and deployment pipelines that turn every merge into a repeatable release instead of a manual, error-prone procedure.

3. Lesson narrative

Deployment is part of the architecture

Hosting changes how identity, WebSockets, logging, configuration, file storage, health checks, and scaling behave. Advanced developers design for the target environment early enough to avoid surprises.

Session-aware scaling

Wisej.NET applications maintain rich server-side state. In load-balanced environments, the infrastructure must route a session consistently or use a deployment model that preserves session behavior. Health checks help route away from unhealthy nodes.

Choosing a deployment target

A Wisej.NET application runs anywhere ASP.NET Core runs, which means the choice is driven by the organization rather than the framework. IIS on Windows fits teams with existing Windows operations, application pool management, and Windows authentication. Kestrel behind an NGINX or Apache reverse proxy fits Linux hosts and gives the proxy responsibility for TLS termination and WebSocket forwarding. Azure App Service and AWS equivalents remove server management but add platform-specific configuration, scaling rules, and logging integration.

Containers change the unit of deployment rather than the host. A Docker image bundles the application and its runtime so that the same artifact runs on a developer machine, a test cluster, and production. The image must still be hosted somewhere, and that host still needs the WebSocket, health check, and session affinity settings the application relies on.

Whatever the target, verify three things before committing: WebSockets pass through every layer between browser and application, the health check endpoint is reachable by the infrastructure, and the application sees the real client address and scheme through the proxy headers.

Load balancing a session-aware application

Every Wisej.NET session lives in the memory of one server process. A load balancer that spreads requests across nodes without affinity would send a user's second request to a node that has never heard of them, and the application would start a new session. Sticky sessions, where the balancer routes all traffic from one client to the same node, are therefore a requirement and not an optimization. The lab's load-balancing notes state this explicitly for the operations team.

Real-time features add a second constraint. Server push relies on a WebSocket that stays open between the browser and the node holding the session, so the balancer and any proxy must keep long-lived connections alive and must not time them out as idle. Scaling out therefore means adding nodes that each hold a share of the sessions, and scaling in means draining a node gracefully so its sessions can end rather than being cut off.

Health checks feed the balancer. In the lab's failure path, node B fails its check, the balancer routes new sessions away from it, and the users on node A never notice. Document what happens to the sessions that were already on node B, because that is the question operations will ask first.

Pipelines, smoke tests, rollback, and runbooks

A release pipeline turns every merge into the same sequence: build, run tests, produce a versioned artifact, deploy it to a staging environment, run smoke tests, and promote it. The value is repetition. When every release follows the same path, a failure is a difference to investigate rather than a mystery. The artifact is versioned, so the release dashboard can show that 2.4.2 is deploying and 2.4.1 is what ran before.

Smoke tests are the few checks that prove the deployed application is alive: the login page renders, a session starts, the health check returns healthy, a known query returns rows, and a WebSocket connects. They run in minutes and they run against the real environment. The rollback plan is what happens when they fail. It names the previous artifact, the command that redeploys it, the person who may invoke it, and how to confirm it worked.

The runbook collects all of this into the document an on-call engineer reads at two in the morning. The lab's runbook ends with rollback, because a release procedure that cannot be reversed is not finished.

Configuration as a contract

The same build must run in development, test, and production, so nothing environment-specific can live in the code. Connection strings, identity provider URLs, storage accounts, feature flags, log levels, and allowed hosts are configuration, supplied by the environment and read at startup. The lab's environment configuration table lists every such setting, its source in each environment, and who owns changing it.

Wisej.NET reads its own settings from `Default.json`, and health check behavior from `HealthCheck.json`. Application settings follow the ASP.NET Core configuration model, so an environment variable or a mounted secrets file can override a value from a file without touching the image. Secrets never appear in the configuration files that are committed. They are injected at deployment time from a vault or the platform's secret store.

- Can the same artifact be deployed to every environment with only configuration changing.
- Does every setting in the table have an owner and a source per environment.
- Would a leaked repository expose any secret.



Advanced production habit

Do not stop when the screen looks correct. Explain state ownership, security implications, failure paths, and production behavior before considering the module complete.



Learning objectives

By the end of this module you'll be able to:

- ✓ Choose deployment targets for IIS, Kestrel/reverse proxy, Azure, AWS, and containers.
- ✓ Prepare load-balancing notes for session-aware real-time applications.
- ✓ Create release pipelines, smoke tests, rollback plans, and runbooks.
- ✓ Separate environment configuration from application code.

Lab / Exam Guide

What you'll build in the hands-on lab, the suggested approach, and the required deliverables.



Hands-on lab

Create a deployment package for EnterpriseOps with environment-specific configuration, HealthCheck.json, smoke test checklist, sample container notes, reverse-proxy notes, sticky-session/load-balancer guidance, and rollback runbook.

Suggested approach

- ✓ Read the production problem and identify the architecture risk before touching the UI.
- ✓ Create or update the relevant service, command, model, or component boundary first.
- ✓ Build the Wisej.NET screen or component using the established project standards.
- ✓ Add a failure-path demonstration, not only the happy path.
- ✓ Record what changed in the learning log and update the relevant checklist.

Required deliverables

- ✓ Deployment architecture diagram
- ✓ Environment configuration table
- ✓ HealthCheck.json
- ✓ Release runbook
- ✓ Rollback and smoke test checklist

Student review questions

- ✓ What requires sticky sessions?
- ✓ Where are secrets stored?
- ✓ How do operations know a node is unhealthy?



Advanced production habit

Do not stop when the screen looks correct. Explain state ownership, security implications, failure paths, and production behavior before considering the module complete.