

Cloud, Container, Load Balancing und Release Engineering

Lektionsleitfaden · Lab- und Prüfungsleitfaden

● LearnWisej

Lektionsleitfaden

IIS, Kestrel, Reverse Proxys, Container, Sticky Sessions, Health Checks, CI/CD, Runbooks. Was das für ein Wisej.NET-Team auf Expertenniveau bedeutet – und wie Sie dieses Modul lesen.



Fokus: Produktivbetrieb

IIS, Kestrel, Reverse Proxys, Container, Sticky Sessions, Health Checks, CI/CD, Runbooks.

1. Das Problem im Produktivbetrieb

Anspruchsvolle Wisej.NET-Anwendungen müssen die Entwicklerrechner verlassen und in reproduzierbare Umgebungen umziehen – mit disziplinierter Konfiguration, Health Checks, Load Balancing, Rollback und klarer Verantwortung im Betrieb.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
IIS	Der Windows-Webserver, der in vielen Unternehmen Wisej.NET-Anwendungen hostet. Anwendungspools, Recycling-Regeln und die WebSocket-Unterstützung müssen konfiguriert werden.
Kestrel	Der plattformübergreifende Webserver von ASP.NET Core, der eine Wisej.NET-Anwendung direkt ausführt – im Produktivbetrieb meist hinter einem Reverse Proxy.
NGINX/Apache als Reverse Proxy	Ein vorgelagerter Server, der TLS terminiert, WebSocket-Verkehr weiterleitet und Anfragen an eine oder mehrere Anwendungsinstanzen verteilt.
Azure/AWS	Cloud-Plattformen, die die Anwendung als App Service, Container oder virtuelle Maschine hosten, jeweils mit eigenem Modell für Skalierung und Konfiguration.
Docker	Verpackt die Anwendung und ihre Laufzeitumgebung in ein Image, sodass derselbe Build auf einem Entwicklerrechner, einem Testserver und in der Cloud identisch läuft.
HealthCheck.json	Die Health-Check-Konfiguration von Wisej.NET, mit der Load Balancer und Orchestratoren prüfen können, ob eine Instanz bereit ist, Sessions zu bedienen.
Sticky Sessions	Affinität im Load Balancer, die jede Anfrage eines Benutzers an dieselbe Serverinstanz leitet. Sie ist zwingend nötig, weil der Session-Zustand im Serverspeicher liegt.
CI/CD	Automatisierte Pipelines für Build, Test und Deployment, die jeden Merge in ein reproduzierbares Release verwandeln statt in einen manuellen, fehleranfälligen Vorgang.

3. Der rote Faden der Lektion

Deployment ist Teil der Architektur

Das Hosting verändert, wie sich Identität, WebSockets, Logging, Konfiguration, Dateispeicher, Health Checks und Skalierung verhalten. Erfahrene Entwickler planen früh genug für die Zielumgebung, um Überraschungen zu vermeiden.

Skalierung mit Blick auf Sessions

Wisej.NET-Anwendungen halten einen umfangreichen Zustand auf dem Server. In Umgebungen mit Load Balancing muss die Infrastruktur eine Session konsistent an denselben Knoten leiten oder ein Deployment-Modell verwenden, das das Session-Verhalten erhält. Health Checks helfen, Verkehr von ungesunden Knoten wegzuleiten.

Ein Deployment-Ziel auswählen

Eine Wisej.NET-Anwendung läuft überall dort, wo ASP.NET Core läuft. Die Wahl hängt deshalb von der Organisation ab, nicht vom Framework. IIS unter Windows passt zu Teams mit bestehendem Windows-Betrieb, Verwaltung von Anwendungspools und Windows-Authentifizierung. Kestrel hinter einem NGINX- oder Apache-Reverse-Proxy passt zu Linux-Hosts und überträgt dem Proxy die Verantwortung für TLS-Terminierung und WebSocket-Weiterleitung. Azure App Service und die entsprechenden AWS-Dienste

nehmen Ihnen die Serververwaltung ab, bringen aber plattformspezifische Konfiguration, Skalierungsregeln und Logging-Integration mit.

Container verändern die Einheit des Deployments, nicht den Host. Ein Docker-Image bündelt die Anwendung und ihre Laufzeitumgebung, sodass dasselbe Artefakt auf einem Entwicklerrechner, in einem Testcluster und im Produktivbetrieb läuft. Das Image muss trotzdem irgendwo gehostet werden, und dieser Host braucht weiterhin die Einstellungen für WebSockets, Health Checks und Session-Affinität, auf die sich die Anwendung verlässt.

Prüfen Sie unabhängig vom Ziel drei Dinge, bevor Sie sich festlegen: WebSockets passieren jede Schicht zwischen Browser und Anwendung, der Health-Check-Endpunkt ist für die Infrastruktur erreichbar, und die Anwendung sieht über die Proxy-Header die echte Clientadresse und das echte Schema.

Load Balancing für eine sessionbewusste Anwendung

Jede Wisej.NET-Session lebt im Speicher eines einzelnen Serverprozesses. Ein Load Balancer, der Anfragen ohne Affinität auf Knoten verteilt, würde die zweite Anfrage eines Benutzers an einen Knoten schicken, der ihn nicht kennt, und die Anwendung würde eine neue Session starten. Sticky Sessions, bei denen der Balancer den gesamten Verkehr eines Clients an denselben Knoten leitet, sind deshalb eine Voraussetzung und keine Optimierung. Die Load-Balancing-Notizen im Lab halten das für das Betriebsteam ausdrücklich fest.

Echtzeitfunktionen bringen eine zweite Einschränkung mit. Server-Push beruht auf einem WebSocket, der zwischen dem Browser und dem Knoten mit der Session offen bleibt. Der Balancer und jeder Proxy müssen langlebige Verbindungen daher aufrechterhalten und dürfen sie nicht wegen Inaktivität trennen.

Hochskalieren bedeutet also, Knoten hinzuzufügen, die jeweils einen Teil der Sessions übernehmen, und Herunterskalieren bedeutet, einen Knoten geordnet leerlaufen zu lassen, damit seine Sessions enden können, statt abgeschnitten zu werden.

Health Checks versorgen den Balancer mit Informationen. Im Fehlerpfad des Labs schlägt der Check von Knoten B fehl, der Balancer leitet neue Sessions von ihm weg, und die Benutzer auf Knoten A bemerken nichts. Dokumentieren Sie, was mit den Sessions geschieht, die bereits auf Knoten B liefen, denn genau das wird der Betrieb als Erstes fragen.

Pipelines, Smoke-Tests, Rollback und Runbooks

Eine Release-Pipeline verwandelt jeden Merge in dieselbe Abfolge: bauen, Tests ausführen, ein versioniertes Artefakt erzeugen, es in eine Staging-Umgebung deployen, Smoke-Tests ausführen und es freigeben. Der Wert liegt in der Wiederholung. Wenn jedes Release denselben Weg nimmt, ist ein Fehler eine Abweichung, die man untersucht, und kein Rätsel. Das Artefakt ist versioniert, sodass das Release-Dashboard zeigen kann, dass gerade 2.4.2 ausgerollt wird und vorher 2.4.1 lief.

Smoke-Tests sind die wenigen Prüfungen, die belegen, dass die deployte Anwendung lebt: Die Anmeldeseite wird gerendert, eine Session startet, der Health Check meldet „healthy“, eine bekannte Abfrage liefert Zeilen und ein WebSocket verbindet sich. Sie laufen in wenigen Minuten und gegen die echte Umgebung. Der Rollback-Plan legt fest, was passiert, wenn sie fehlschlagen. Er nennt das vorherige Artefakt, den Befehl, der es erneut deployt, die Person, die ihn auslösen darf, und wie man bestätigt, dass er funktioniert hat.

Das Runbook fasst all das in dem Dokument zusammen, das ein Engineer in Rufbereitschaft um zwei Uhr nachts liest. Das Runbook im Lab endet mit dem Rollback, denn ein Release-Verfahren, das sich nicht rückgängig machen lässt, ist nicht fertig.

Konfiguration als Vertrag

Derselbe Build muss in Entwicklung, Test und Produktivbetrieb laufen, deshalb darf nichts Umgebungsspezifisches im Code stehen. Connection Strings, URLs des Identity Providers, Speicherkonten, Feature-Flags, Log-Level und zulässige Hosts sind Konfiguration: Die Umgebung stellt sie bereit, und sie werden beim Start gelesen. Die Tabelle der Umgebungskonfiguration im Lab listet jede dieser Einstellungen auf, ihre Quelle in jeder Umgebung und wer für Änderungen verantwortlich ist.

Wisej.NET liest seine eigenen Einstellungen aus `Default.json` und das Verhalten des Health Checks aus `HealthCheck.json`. Anwendungseinstellungen folgen dem Konfigurationsmodell von ASP.NET Core, sodass eine Umgebungsvariable oder eine eingebundene Secrets-Datei einen Wert aus einer Datei überschreiben kann, ohne das Image anzufassen. Secrets erscheinen nie in den Konfigurationsdateien, die eingecheckt werden. Sie werden beim Deployment aus einem Vault oder dem Secret Store der Plattform eingespielt.

- Lässt sich dasselbe Artefakt in jede Umgebung deployen, wobei sich nur die Konfiguration ändert?
- Hat jede Einstellung in der Tabelle einen Verantwortlichen und eine Quelle pro Umgebung?
- Würde ein geleaktes Repository irgendein Secret preisgeben?



Expertenregel für den Produktivbetrieb

Hören Sie nicht auf, sobald der Bildschirm korrekt aussieht. Erklären Sie, wem der Zustand gehört, welche Sicherheitsfolgen es gibt, welche Fehlerpfade existieren und wie sich das System im Produktivbetrieb verhält, bevor Sie das Modul als abgeschlossen betrachten.



Lernziele

Am Ende dieses Moduls können Sie:

- ✓ Deployment-Ziele für IIS, Kestrel/Reverse Proxy, Azure, AWS und Container auswählen.
- ✓ Load-Balancing-Notizen für sessionbewusste Echtzeitanwendungen vorbereiten.
- ✓ Release-Pipelines, Smoke-Tests, Rollback-Pläne und Runbooks erstellen.
- ✓ die Umgebungskonfiguration vom Anwendungscode trennen.

Lab- und Prüfungsleitfaden

Was Sie im Praxis-Lab bauen, das vorgeschlagene Vorgehen und die geforderten Ergebnisse.



Praxis-Lab

Erstellen Sie ein Deployment-Paket für EnterpriseOps mit umgebungsspezifischer Konfiguration, HealthCheck.json, Smoke-Test-Checkliste, Beispielnutzen zum Container, Notizen zum Reverse Proxy, Hinweisen zu Sticky Sessions und Load Balancer sowie einem Rollback-Runbook.

Vorgeschlagenes Vorgehen

- ✓ Lesen Sie das Problem im Produktivbetrieb und benennen Sie das Architekturrisiko, bevor Sie die UI anfassen.
- ✓ Legen Sie zuerst die betroffene Grenze für Service, Command, Modell oder Komponente an oder aktualisieren Sie sie.
- ✓ Erstellen Sie den Wisej.NET-Bildschirm oder die Komponente nach den etablierten Projektstandards.
- ✓ Demonstrieren Sie auch einen Fehlerpfad, nicht nur den Happy Path.
- ✓ Halten Sie die Änderungen im Lernprotokoll fest und aktualisieren Sie die zugehörige Checkliste.

Geforderte Ergebnisse

- ✓ Diagramm der Deployment-Architektur
- ✓ Tabelle der Umgebungsconfiguration
- ✓ HealthCheck.json
- ✓ Release-Runbook
- ✓ Checkliste für Rollback und Smoke-Test

Wiederholungsfragen

- ✓ Was erfordert Sticky Sessions?
- ✓ Wo werden Secrets gespeichert?
- ✓ Woran erkennt der Betrieb, dass ein Knoten nicht mehr gesund ist?



Expertenregel für den Produktivbetrieb

Hören Sie nicht auf, sobald der Bildschirm korrekt aussieht. Erklären Sie, wem der Zustand gehört, welche Sicherheitsfolgen es gibt, welche Fehlerpfade existieren und wie sich das System im Produktivbetrieb verhält, bevor Sie das Modul als abgeschlossen betrachten.