Ordivis Belegung

BETRIEB

Betriebshandbuch — Installation und Serverbetrieb

Auslieferungspaket, Datenbank, Dienst, Aktualisierung und Bauen aus dem Quelltext

ALPHA · Fassung 0.9.0

Handbuch-Fassung 1.0 · Stand 29.08.2026

Grams IT · Produktreihe Ordivis · .NET 10 · PostgreSQL · Exchange

ID-50 Installation

Betriebshandbuch — Installation und Serverbetrieb

Achtung: Alpha-Fassung. Ordivis Belegung steht in der Fassung 0.9.0 und ist eine **Alpha-Fassung**. Die Anwendung wird laufend weitergebaut: **Masken, Menüwege, Bezeichnungen, Konfigurationsschlüssel und Abläufe können sich jederzeit und ohne Vorankündigung ändern** — und mit ihnen die Angaben in diesem Handbuch. Dieses Kapitel beschreibt den Stand vom 29.08.2026. Weicht die Anwendung von einer Beschreibung ab, gilt die Anwendung und nicht dieses Blatt; melden Sie die Abweichung. Für den Echtbetrieb ist diese Fassung **nicht freigegeben**.

Für den IT-Betrieb. Der Weg von der leeren Maschine bis zum laufenden Dienst.

INHALT

1 · Das Auslieferungspaket

Auf der Kundenmaschine wird **nichts gebaut**. Das Paket (`ordivis-belegung-<Fassung>.zip`) enthält alles Übersetzte:

Im Paket	Inhalt
`anwendung/`	die übersetzte Fachanwendung — läuft **intern**, nie im offenen Netz
`oeffentlich/`	die übersetzte Oberfläche, statische Dateien
`schema.sql`	das Datenbankschema, einspielbar auch über eine bestehende Datenbank
`vorlagen/`	Nutzungsvertrag und Abrechnung als Vorlage
`LIESMICH.txt`	Fassung, Voraussetzungen, die vier Schritte

Dasselbe Paket entsteht am Arbeitsplatz mit:

```
pwsh -File tools\Baue-Auslieferung.ps1
```

Framework-abhängig, nicht eigenständig. Das Paket braucht auf der Zielmaschine die **ASP.NET-Core-Laufzeit 10** — kein SDK.

Warum nicht eigenständig: Ein eigenständiges Paket bräuchte die Laufzeit nicht, aber dann käme **jede Sicherheitsberichtigung von .NET nur noch über ein neues Paket vom Hersteller**. So kommt sie von der Plattform, wie bei jeder anderen Anwendung. Dasselbe Paket läuft auf Windows wie auf Linux.

1a · Das Windows-Setup

⚙️ **Am Server:** `Ordivis-Belegung-Setup-<Fassung>.exe` · 🔑 Administrator

Der kürzeste Weg auf eine Windows-Maschine. Das Setup tut genau das, was in §2 von Hand steht — und zwar in derselben Reihenfolge:

Schritt	Was geschieht
1	**ASP.NET-Core-Laufzeit 10** prüfen und beschaffen, wenn sie fehlt
2	**PostgreSQL 18** installieren — *nur, wenn im Assistenten gewählt*
3	**Datenbank und Rolle** anlegen, `schema.sql` einspielen
4	`appsettings.Production.json` schreiben — aus den Angaben des Assistenten
5	**Windows-Dienst** anlegen, Port in der Firewall freigeben, starten

✅ **Ergebnis:** Der Dienst `OrdivisBelegung` läuft, und das Setup hat das **Lebenszeichen abgefragt**, bevor es „fertig" meldet.

Warum das Lebenszeichen und nicht der Dienstzustand. Ein Dienst im Zustand *Running*, dessen Anwendung beim Start an einer fehlenden Pflichtangabe abgebrochen ist, sieht genauso aus wie einer, der arbeitet — und der Betrieb hielte eine kaputte Anlage für eingerichtet.

Was der Assistent fragt

Seite	Angaben
Träger	Name, Straße, PLZ, Ort — alle vier sind Pflicht (ID-40 §2)
Anwendung	Port (Vorgabe 5080), öffentliche Adresse, Funktionspostfach
Datenbankserver	vorhandenen verwenden **oder** PostgreSQL 18 mitinstallieren
Datenbank	Rechner, Port, Verwaltungskonto, Rolle für die Anwendung
Exchange	Endpunkt und Dienstkonto — **darf leer bleiben** und später über die Maske kommen (ID-42 §5)

Die vier Trägerangaben werden im Assistenten erzwungen. Sie hier abzufragen und dann durchzuwinken hieße, das Setup als gelungen zu melden und einen Dienst zu hinterlassen, der gar nicht hochkommt.

Der Datenbankserver — die eine Wahl, die Sie treffen müssen

Wahl	Wann
Vorhandenen Server verwenden	Im Haus läuft schon eine PostgreSQL. Sicherung, Pflege und Überwachung sind dort geregelt — das ist der bessere Weg
PostgreSQL 18 mitinstallieren	Es gibt keine. Rund 350 MB werden geladen, die Installation dauert ein bis zwei Minuten

Die Vorauswahl richtet sich danach, was das Setup vorfindet.

Achtung: Auf einem Rechner mit vorhandener PostgreSQL wird KEINE zweite installiert — unabhängig davon, was im Assistenten steht. Der Schritt erkennt die vorhandene Instanz und steigt aus; das ist kein Fehlschlag, sondern die richtige Antwort.

Warum das ein Riegel und nicht nur ein Hinweis ist: Zwei Instanzen auf einer Maschine fallen erst auf, wenn die Sicherung die falsche mitnimmt.

Achtung: Bei „mitinstallieren" bedeutet das Kennwortfeld etwas anderes. Es ist dann **kein vorhandenes Kennwort**, sondern das, welches der frische Superuser `postgres` **bekommt**. Der Assistent sagt das auf der Seite — wer es überliest, tippt sein altes ein und wundert sich, dass es nicht mehr gilt.

Was mitinstalliert wird und was nicht: Server und `psql`. **pgAdmin und StackBuilder bleiben weg** — sie machen den Großteil der rund 1 GB und der Installationszeit aus, und ein Dienst braucht keine Oberfläche. Der Cluster kommt auf das Laufwerk mit dem **meisten freien Platz**: Er wächst, und die Systemplatte ist selten die größte.

> **Tipp: Auf einem Server ohne Weg ins offene Netz** legen Sie das Installationsprogramm von PostgreSQL neben die Setup-Datei, in einen Ordner `vorrat`. Dann lädt das Setup nichts. Einzelheiten in `installer\vorrat\LIESMICH.txt`.

Was das Setup NICHT tut

> **Achtung: Es überschreibt keine bestehende `appsettings.Production.json`.** Sie gehört der Kommune. Gibt es sie schon, legt das Setup die neue Fassung als `appsettings.Production.json.neu` daneben — zum Vergleichen, nicht zum Ersetzen. Dasselbe gilt für die Vorlagen unter `vorlagen\`.
>
> **Und es entfernt PostgreSQL bei der Deinstallation nicht** — auch dann nicht, wenn es ihn selbst installiert hat. Dort liegen die Daten.

Was die Deinstallation stehen lässt

Entfernt werden das Programmverzeichnis, der Dienst und die Freigabe in der Firewall. **Erhalten bleiben** die Datenbank mit allen Vorgängen und die Konfiguration — eine Neuinstallation nutzt beides unverändert weiter.

Das Setup bauen

```
pwsh -File installer\Baue-Setup.ps1
```

Es baut zuerst dieselbe Nutzlast wie das ZIP-Paket (§1) und packt sie dann.

> **Die Nutzlast wird nicht eigens für das Setup gebaut.** Setup und ZIP tragen damit bitgleich dasselbe — sonst gäbe es zwei Auslieferungswege, die auseinanderlaufen können, und der Supportfall begänne mit der Frage, welchen von beiden der Kunde genommen hat.

2 · Installation in vier Schritten

> ⚙️ **Am Server:** Konsole · 🔑 Rechte zum Anlegen von Datenbank und Dienst

🔒 **Pflichtreihenfolge**

Schritt 1 — Datenbank anlegen

```
CREATE DATABASE buchung;
```

Achtung: Die Anwendung lässt nur die Namen `buchung` und `buchung_test` zu. Liegen fremde Datenbanken auf derselben Instanz, merkt eine Migration in die falsche niemand sofort — und zurück kommt man nur über die Sicherung. Ein anderer Name bricht den Start mit einer Meldung ab, statt loszulaufen.

Schritt 2 — Schema einspielen

```
psql -d buchung -f schema.sql
```

Derselbe Befehl für die erste Einrichtung und für jede Aktualisierung. Das Skript ist *idempotent* erzeugt und prüft je Migration, ob sie schon eingespielt ist.

Warum die Anwendung nicht selbst migriert. Sie könnte es beim Start, und für den Kunden wäre das bequemer. Sie tut es nicht: Eine Änderung an einer Datenbank, die Vorgänge von Bürgerinnen hält, soll jemand **lesen** können, bevor sie läuft — und zwei gleichzeitig startende Anwendungen versuchten sie zweimal.

Schritt 3 — Konfiguration füllen

`appsettings.Production.json` und die Geheimnisse (ID-40).

Mindestens: `Traeger:*`, `ConnectionStrings:Buchung`, `Exchange:*` mit Kennwort, `Meldungen:Funktionspostfach`, `Sicherheit:Datenschluessel`.

Schritt 4 — Starten

```
dotnet anwendung/Ordivis.Belegung.Api.dll
```

✅ **Ergebnis:** Die Startzeilen melden Verwaltungsbereich, Exchange und Warteschlange (ID-40 §1). Danach: Ersteinrichtung (ID-41 §2) und Erstbefüllung (ID-40 §19).

3 · Als Dienst einrichten

Linux (systemd)

```
[Unit]
Description=Ordivis Belegung
After=network-online.target postgresql.service

[Service]
Type=notify
WorkingDirectory=/opt/ordivis-belegung
ExecStart=/usr/bin/dotnet /opt/ordivis-belegung/anwendung/Ordivis.Belegung.Api.dll
User=ordivis
Environment=ASPNETCORE_ENVIRONMENT=Production
Environment=Sicherheit__Einrichtungsverzeichnis=/var/lib/ordivis-belegung
StateDirectory=ordivis-belegung
ProtectSystem=strict
ProtectHome=true
PrivateTmp=true
NoNewPrivileges=true
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

Wichtig: `ProtectSystem=strict` **macht** `/opt` **schreibgeschützt.** Dann braucht die Ersteinrichtung ein beschreibbares Verzeichnis — `Sicherheit__Einrichtungsverzeichnis` zeigt oben deshalb auf das Zustandsverzeichnis. Ohne diese Angabe startet die Anwendung trotzdem; der Einrichtungsschlüssel steht dann nur im Protokoll und **gilt nur bis zum nächsten Dienstneustart** (ID-41 §2).

Die Geheimnisse gehören nicht in die Einheitendatei, sondern in eine `EnvironmentFile=` mit `0600` oder in den Secret Store der Plattform.

Windows

Als Windows-Dienst über `sc.exe create` oder einen Dienstwrapper. Dasselbe gilt: Geheimnisse über Umgebungsvariablen des Dienstkontos, nicht in einer Datei neben der Anwendung.

4 · Welche Fassung läuft gerade?

```
GET /verwaltung/betrieb/fassung
```

Hinter der Anmeldung. Nennt Fassung, Baustand (mit Commit) und die Laufzeit.

> **Achtung: Bewusst nicht unter `/gesundheit`.** Eine Fassungsnummer im offenen Netz ist eine Auskunft für jeden, der wissen will, welche Lücken sie hat.

Das Lebenszeichen ist bewusst wortkarg:

```
GET /gesundheit   →  {"zustand":"läuft"}
```

5 · Aktualisieren

🔒 **Pflichtreihenfolge**

1. **Sichern** (ID-53) — Datenbank, Konfiguration, Schlüssel, Bilder.
2. Dienst **anhalten**.
3. `anwendung/` und `oeffentlich/` aus dem neuen Paket ersetzen. `appsettings.Production.json` **NICHT überschreiben** — sie gehört der Kommune und liegt im Paket gar nicht erst.
4. `psql -d buchung -f schema.sql` — derselbe Befehl wie bei der Erstinstallation.
5. Dienst **starten**, Startzeilen lesen.
6. `/verwaltung/betrieb/fassung` aufrufen und die Fassung prüfen.
7. *Pflege* ▸ *Störungsbild* aufrufen: Der Spiegel ist zunächst älter und legt sich mit dem nächsten Abgleich.

> **Achtung: Beim Aktualisieren einmal nachsehen, ob `Lizenz:Aktivierungsdienst` ausdrücklich in der eigenen Konfiguration steht** (ID-43 §6). Eine veraltete Adresse fällt nicht auf: Die Online-Bestätigung ist beratend und nie sperrend.
>
> **Prüfen Sie nach jeder Auslieferung die Vorlagen.** Kamen neue Platzhalter dazu, tragen die alten Vorlagen sie nicht — der Vertrag ist dann inhaltlich unvollständig, aber technisch fehlerfrei (ID-25).

6 · Aus dem Quelltext bauen

Nur auf einer Entwicklungsmaschine.

```
dotnet build
dotnet test
```

Die Integrationstests laufen gegen den **echten** Exchange und eine echte PostgreSQL-Datenbank. Fehlen die Zugangsdaten, überspringen sie sich selbst — der Bau wird auf einem Rechner ohne Netz nicht rot.

> **Achtung: Ein grüner Bauserver heißt nicht, dass die Exchange-Anbindung geprüft wurde.** Sie überspringt sich dort mangels Zugangsdaten. Dafür gibt es `tools\Test-EwsDeltaAbgleich.ps1` und den Lauf am Arbeitsplatz.

Anwendung starten:

```
dotnet run --project src/Ordivis.Belegung.Api
```

Oberfläche bauen:

```
npm --prefix src/Ordivis.Belegung.Web ci
npm --prefix src/Ordivis.Belegung.Web run build
```

Schema aus dem Quelltext einspielen:

```
dotnet ef database update --project src/Ordivis.Belegung.Infrastruktur --startup-project src
```

Die Verbindungszeichenfolge liest `dotnet ef` aus `BUCHUNG_DB` oder aus den Benutzergeheimnissen — **nie** aus einer Datei im Projektverzeichnis.

Die Abdeckung hält eine Schwelle

`dotnet test` misst die Abdeckung der **Domäne** und bricht ab, wenn sie unter 85 % Zeilen oder 80 % Zweige fällt.

> **Die Schwelle ist eine Sperrklinke, kein Ziel.** Sie fängt einen Rückschritt, sie fordert keinen Fortschritt. Wer sie hebt, tut das nach einer Messung — und nicht, weil sie beim ersten Widerstand im Weg stand.

Stolperstein: eine maschinenweite `ASPNETCORE_ENVIRONMENT`

> **Achtung:** Steht auf der Maschine `ASPNETCORE_ENVIRONMENT=Production` als System- oder Benutzervariable, **überstimmt sie die Startprofile**. Die Anwendung startet dann in `Production`, lädt keine Benutzergeheimnisse und bricht mit „Keine Verbindungszeichenfolge gesetzt" ab — **obwohl das Geheimnis gesetzt ist.**

```
ASPNETCORE_ENVIRONMENT=Development \
ConnectionStrings__Buchung="…" \
EWS_KENNWORT="…" \
dotnet run --project src/Ordivis.Belegung.Api --no-launch-profile
```

7 · Aufbau des Quelltextes

Projekt	Inhalt
`src/Ordivis.Belegung.Domaene`	Entitäten und Regeln. **Ohne jeden Verweis**
`src/Ordivis.Belegung.Anwendung`	Anwendungsfälle, Belegungsquelle, Spiegelabgleich
`src/Ordivis.Belegung.Infrastruktur`	PostgreSQL über EF Core, EWS-Anbindung

Projekt	Inhalt
`src/Ordivis.Belegung.Api`	ASP.NET Core, Endpunkte, Dienste
`src/Ordivis.Belegung.Web`	Angular-Oberfläche
`tests/Ordivis.Belegung.Tests`	Domänenregeln, Schichtenschnitt, Integrationstests

Der Schichtenschnitt wird **geprüft, nicht vereinbart**.

8 · Häufige Fehlerbilder beim Start

Meldung	Ursache
„Keine Verbindungszeichenfolge gesetzt"	Geheimnis fehlt — oder §6, Stolperstein
„Datenbankname … nicht zugelassen"	Nicht `buchung` / `buchung_test` (§2)
Start bricht mit fehlender Trägerangabe ab	`Traeger:*` unvollständig (ID-40 §2)
„Windows-Anmeldung ohne Gruppe"	Mindestens eine AD-Gruppe setzen (ID-41 §3)
`Exchange=OHNE Anbindung`	Kennwort oder Endpunkt fehlt (ID-42 §4)
Start bricht mit Schwellenfehler ab	`VeraltetNachMinuten` ≤ `VerzoegertNachMinuten`
`Basic` ohne `https`	Verfahren oder Endpunkt berichtigen

Verwandte Themen

- **ID-02** — Systemvoraussetzungen und Architektur
- **ID-40** — Konfiguration
- **ID-51** — Öffentliche Auslieferung und Webserver
- **ID-53** — Sicherung und Wiederherstellung
- **ID-55** — Prüfliste vor der Freischaltung