Ordivis

TEIL VI · BETRIEB

Betriebshandbuch – Installation & Serverbetrieb

Installation, Update, Sicherung, Härtung und Fehlerbehebung

Produktversion 2026.08.19 · Handbuch-Revision 1.0
Stand 19.08.2026 · Plattform .NET 10 / Windows Server

ID-50 Betrieb

Betriebshandbuch – Installation & Serverbetrieb

Dieses Dokument richtet sich an **IT-Betrieb / Server-Administratoren**. Es beschreibt Installation, Aktualisierung, Sicherung, Härtung und Fehlerbehebung. Architektur und Voraussetzungen stehen in ID-02.

INHALT

1 Deployment-Grundsätze

- **Kein Docker.** Betrieb nativ auf Windows Server.
- Komponenten als **Windows-Dienste**: API und Self-Service-Portal (:8080). Die Netzscans laufen **in der API**; ein eigener Dienst dafür existiert nur im Sonderfall eines entfernten Netzes (Discovery-Collector).
- **PostgreSQL** nativ; **WinUI-Client** unpackaged je Arbeitsplatz.

2 Installation (Reihenfolge)

1. **PostgreSQL** installieren.

   > **Voraussetzung:** Installer mit `--install_runtimes 1` aufrufen, sonst fehlt die VC++-Runtime.

2. **Datenbank & Rolle** anlegen (`infradesk_app`), Rechte vergeben.
3. **API-Dienst** installieren (Setup/Installer), `appsettings` konfigurieren (Connection-String, `CredentialVault:EncryptionKey` als Secret/Umgebungsvariable).
4. **Portal** als Dienst einrichten.
5. **Migrationen** ausführen (siehe Abschnitt 4).
6. **Erststart/Onboarding** durchführen (ID-40).

> **Kein Discovery-Dienst auf dem Server.** Die serverseitigen Scans laufen im API-Dienst selbst; es gibt dafür keinen eigenen Windows-Dienst. Ein separater Dienst existiert nur für den **Collector**, und zwar am entfernten Standort – nicht in der Zentrale (Abschnitt 2.2.2).

> **Wichtig:** Dienste mit `New-Service` anlegen, nicht mit `sc.exe create` – letzteres scheitert unter PowerShell 5.1 still am Pfad-Quoting.

2.1 Server vollständig bereinigen (für eine saubere Neuinstallation)

> **Wichtig – die normale Deinstallation genügt dafür nicht.** Sie entfernt nur die Dienste und Programmdateien; **PostgreSQL und die Ordivis-Datenbank bleiben erhalten** (gewollt, damit ein Update keine Daten verliert). Eine spätere Neuinstallation **verwendet diese Datenbank weiter**: Enthält sie bereits einen Mandanten, gilt das System als eingerichtet – der **Erststart-Assistent erscheint dann nicht**, und die bisherigen Anmeldedaten gelten weiter. Der Installer weist am Ende darauf hin, wenn er eine vorhandene Datenbank übernommen hat. Wer wirklich „von Null" beginnen will, nutzt die folgende Bereinigung.

Für eine echte Neuinstallation „von Null" – oder um Reste eines fehlgeschlagenen Versuchs zu entfernen – gibt es das Skript `cleanup-server.ps1` (im Setup-Paket unter `deploy\inno\`). Es entfernt **vollständig**:

- die Windows-Dienste `InfraDesk.API` , `InfraDesk.Portal` und die PostgreSQL-Dienste,
- führt den **PostgreSQL-Uninstaller** aus,
- löscht die Installations- und Datenverzeichnisse (`…\Ordivis Platform` , `…\PostgreSQL` , `C:\ProgramData\InfraDesk`) inklusive eines individuell gewählten Backup-Verzeichnisses,
- entfernt Firewall-Regeln, Machine-Umgebungsvariablen und die geplante Aufgabe `InfraDesk-DailyBackup` .

Die .NET-Runtime bleibt bewusst erhalten.

> ⚠️ **Unwiderruflich:** Die Ordivis-Datenbank **und alle Sicherungen** werden gelöscht. Vorher bei Bedarf eine Kopie außerhalb der genannten Verzeichnisse ablegen.

Als **Administrator** ausführen (fragt einmal „ja/nein"):

```
powershell -ExecutionPolicy Bypass -File .\cleanup-server.ps1
```

Ohne Rückfrage (z. B. für Automatisierung): zusätzlich `-Force`. Der **Installer erkennt Altbestände auch selbst** und bietet die Bereinigung vor der Installation an – der manuelle Aufruf ist also optional, für einen echten Neustart aber die saubere Variante.

2.2 Installations-Assistent (Bildschirm für Bildschirm)

Der Windows-Installer (`Ordivis-Setup-<Version>.exe` , als **Administrator** ausführen) führt in wenigen Schritten durch die Einrichtung. Alle Passwörter werden zufällig vorbelegt und am Ende in `C:\ProgramData\InfraDesk\setup-credentials.txt` gesichert.

Die Abbildungen 1 bis 3 zeigen die ersten Seiten des Assistenten.

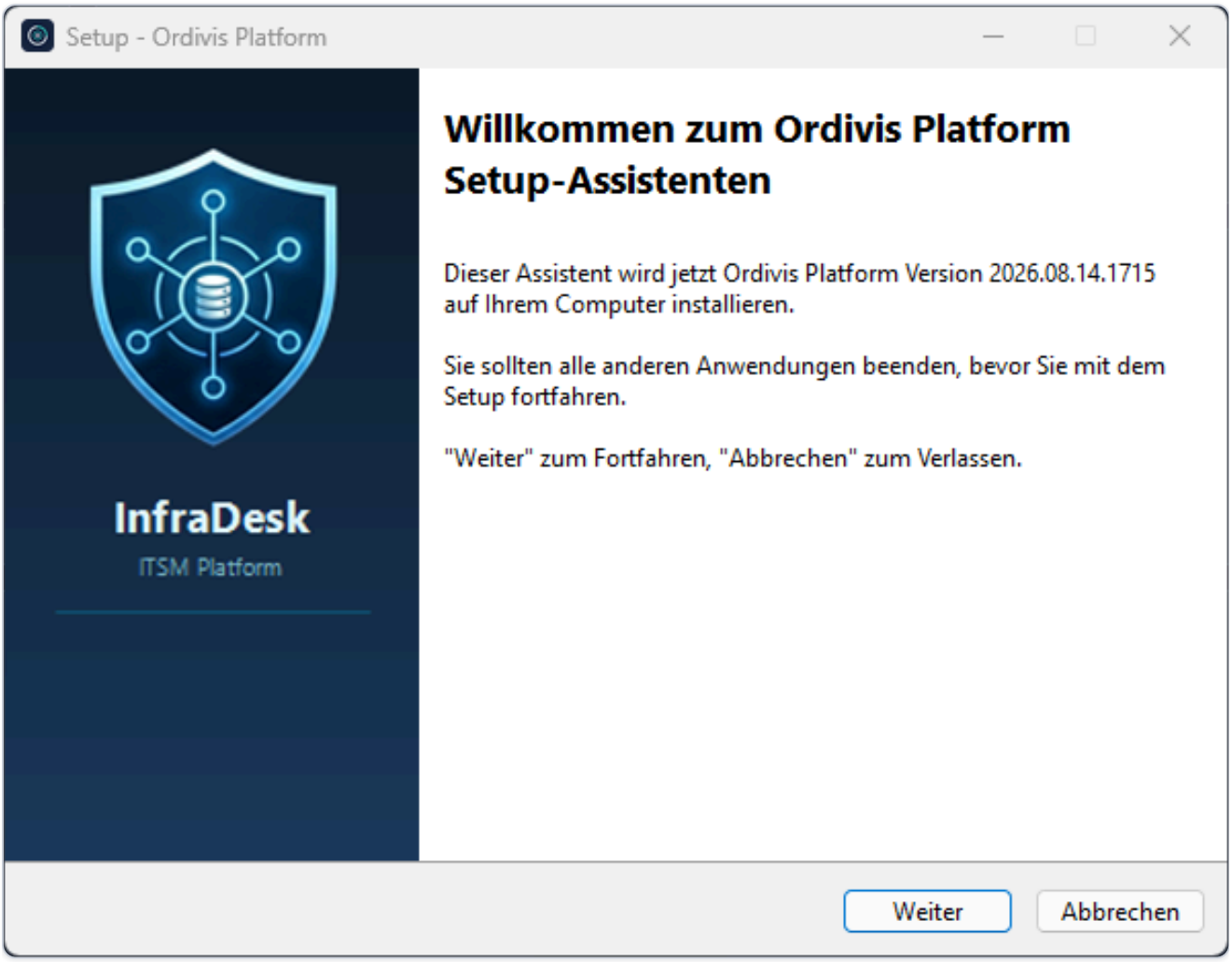

Abb. 1 – Startseite des Assistenten. Es folgen Lizenzvereinbarung und die Auswahl des Installationstyps.

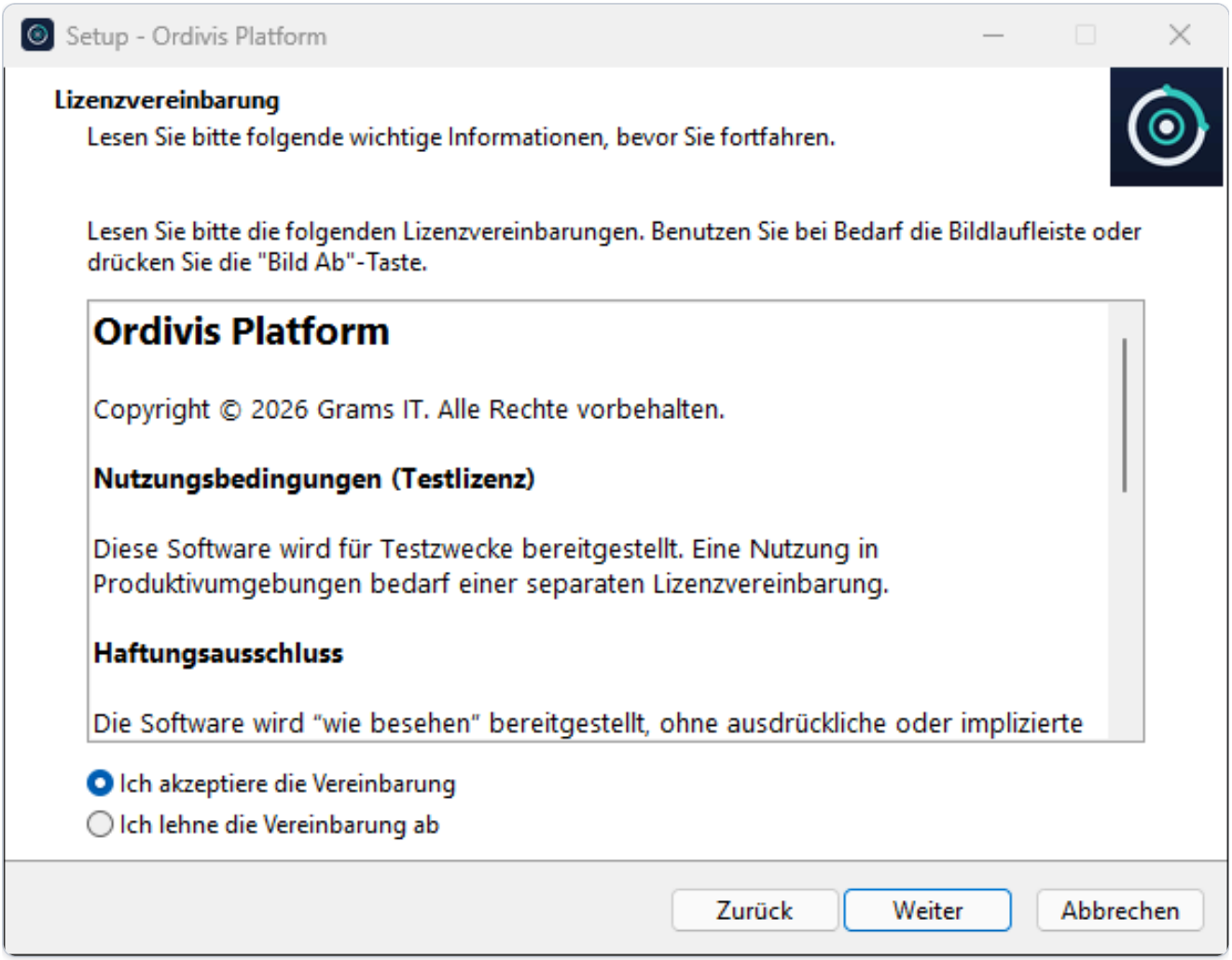

Abb. 2 – **Lizenzvereinbarung**: Zum Fortfahren „Ich akzeptiere die Vereinbarung" wählen.

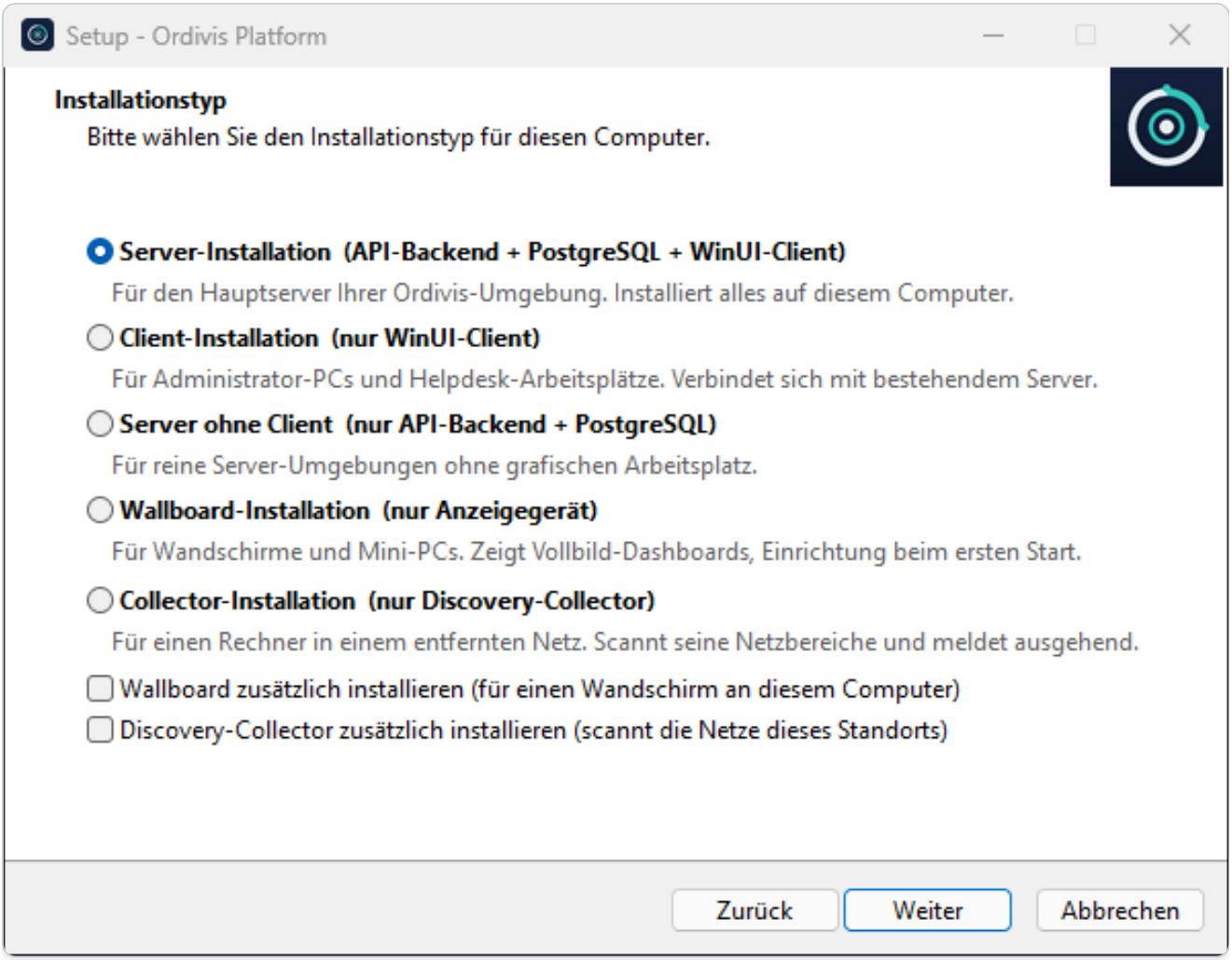

Abb. 3 – **Installationstyp**: *Server* (API + PostgreSQL + Client), reine *Client*-Installation, *Server ohne Client* oder *Wallboard-Installation*. (Die Abbildung zeigt noch den Stand vor 2026.07.28.1912 mit drei Typen.)

Seit Version 2026.07.28.1912 gibt es einen **vierten Installationstyp: Wallboard-Installation**. Er ist für Geräte gedacht, die *nur* als Anzeigebildschirm dienen – ein Rechner am Wandschirm im Teamraum, ohne Arbeitsplatz-Client und ohne Server. Installiert wird dann ausschließlich die Wallboard-Anwendung samt der benötigten Windows-App-Laufzeit.

Soll ein Rechner **beides** sein – Arbeitsplatz oder Server *und* zusätzlich einen Wandschirm bedienen –, setzen Sie bei den Installationstypen *Server* oder *Client* einfach den **Zusatzhaken für das Wallboard**.

> **Hinweis zu älteren Setup-Dateien.** In Setup-Dateien vor 2026.07.28.1912 war das Wallboard zwar enthalten, ließ sich aber nicht auswählen – der Assistent übersprang die Komponentenauswahl und setzte hart nur die drei bekannten Installationstypen. Wer ein Anzeigegerät einrichten will, braucht daher mindestens diese Version. Ebenfalls behoben: Bei einer reinen Wallboard-Installation wird die Windows-App-Laufzeit jetzt mit eingerichtet; ohne sie wäre die Anwendung nach der Installation nicht gestartet.

Seit **2026.07.29** gibt es einen **fünften Installationstyp: Collector-Installation** – der Discovery-Collector für einen Rechner in einem entfernten Netz (siehe Kapitel **Modul Discovery**, Abschnitt 6). Auch er ist als **Zusatzhaken** neben *Server* oder *Client* wählbar, wenn Zentrale und Collector auf derselben Maschine laufen sollen.

Wallboard und Collector sind damit **Zusatzrollen, keine Installationsarten**: Beide lassen sich einzeln hinzunehmen und einzeln weglassen. Wer nur eines von beiden braucht, wählt genau das.

2.2.1 Die drei Installationspakete

Nicht jede Rolle braucht das vollständige Paket:

Paket	Größe	Enthält	Für
`Ordivis-Setup-<Version>.exe`	~630 MB	Server, PostgreSQL, Portal, Client; Wallboard und Collector zuwählbar	die Zentrale
`Ordivis-Wallboard-Setup-<Version>.exe`	~90 MB	nur die Wallboard-Anwendung	Anzeigegeräte
`Ordivis-Collector-Setup-<Version>.exe`	~70 MB	nur den Collector-Dienst	Standort-Rechner in fremden Netzen

Die beiden schlanken Pakete gibt es aus einem handfesten Grund: Einem Kunden oder einer Außenstelle das 630-MB-Paket zu schicken, hieße dort Server, Datenbank und Portal mitzuliefern – Software, die auf einem Anzeigegerät oder einem Scan-Rechner nichts zu suchen hat.

> **Sie lassen sich nebeneinander installieren.** Jedes Paket trägt eine eigene Kennung. Ein schlankes Paket behandelt eine vorhandene Vollinstallation deshalb **nicht** als Upgrade und nimmt sie beim Deinstallieren **nicht** mit.

2.2.2 Collector-Installation (Standort-Rechner)

Der Collector-Installer verlangt **Administratorrechte** – anders als der Wallboard-Installer, der auch ins Benutzerprofil installieren kann. Grund: Der Collector wird als Windows-Dienst registriert und verschlüsselt sein Scan-Kennwort mit dem Maschinenschlüssel; beides geht nicht ohne Erhöhung.

Er fragt zwei Seiten ab:

1. **Verbindung zur Zentrale** – Adresse der Ordivis-API und der **Anmeldeschein** aus der Collector-Verwaltung. Die Zertifikatsprüfung lässt sich abschalten (nur für selbstsignierte Zertifikate) – als ausdrücklicher Haken, nicht als Textfeld: Über diese Verbindung reist ein Zugang.
2. **Anmeldedaten für den Tiefen-Scan** – ein **lesendes** Windows-Konto, optional. Ohne Konto bleibt es beim Netz-Durchlauf.

Nach der Installation läuft der Dienst **InfraDesk.Discovery.Collector** (verzögerter Automatikstart). Was er tut, steht in `C:\ProgramData\InfraDesk\Logs\worker-*.log`; seine Einstellungen in `<Installationsverzeichnis>\collector\appsettings.json`.

> **Der Dienst startet nur, wenn er etwas tun kann.** Ohne Serveradresse wird er angelegt, aber nicht gestartet – ein Dienst, der beim Start sofort „nicht konfiguriert" meldet, sieht im Ereignisprotokoll wie ein Fehler aus und ist doch nur unfertig eingerichtet.

Die Abbildungen 4 bis 12 zeigen die weiteren Seiten in der Reihenfolge, in der sie erscheinen.

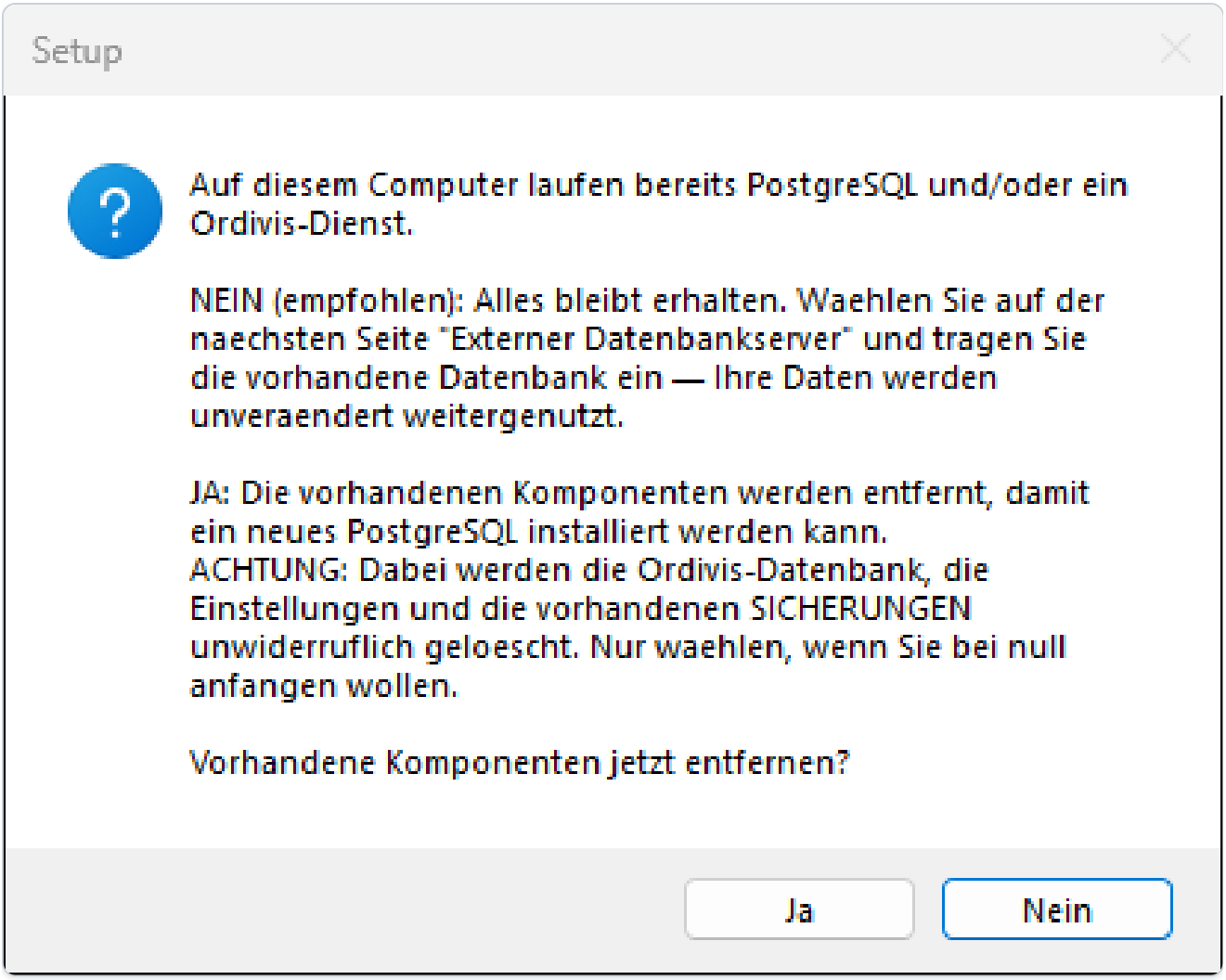

Abb. 4 – Findet das Setup Reste einer früheren Installation, bietet es die **automatische Bereinigung** an (siehe 2.1). Bei einer Erstinstallation erscheint dieser Dialog nicht.

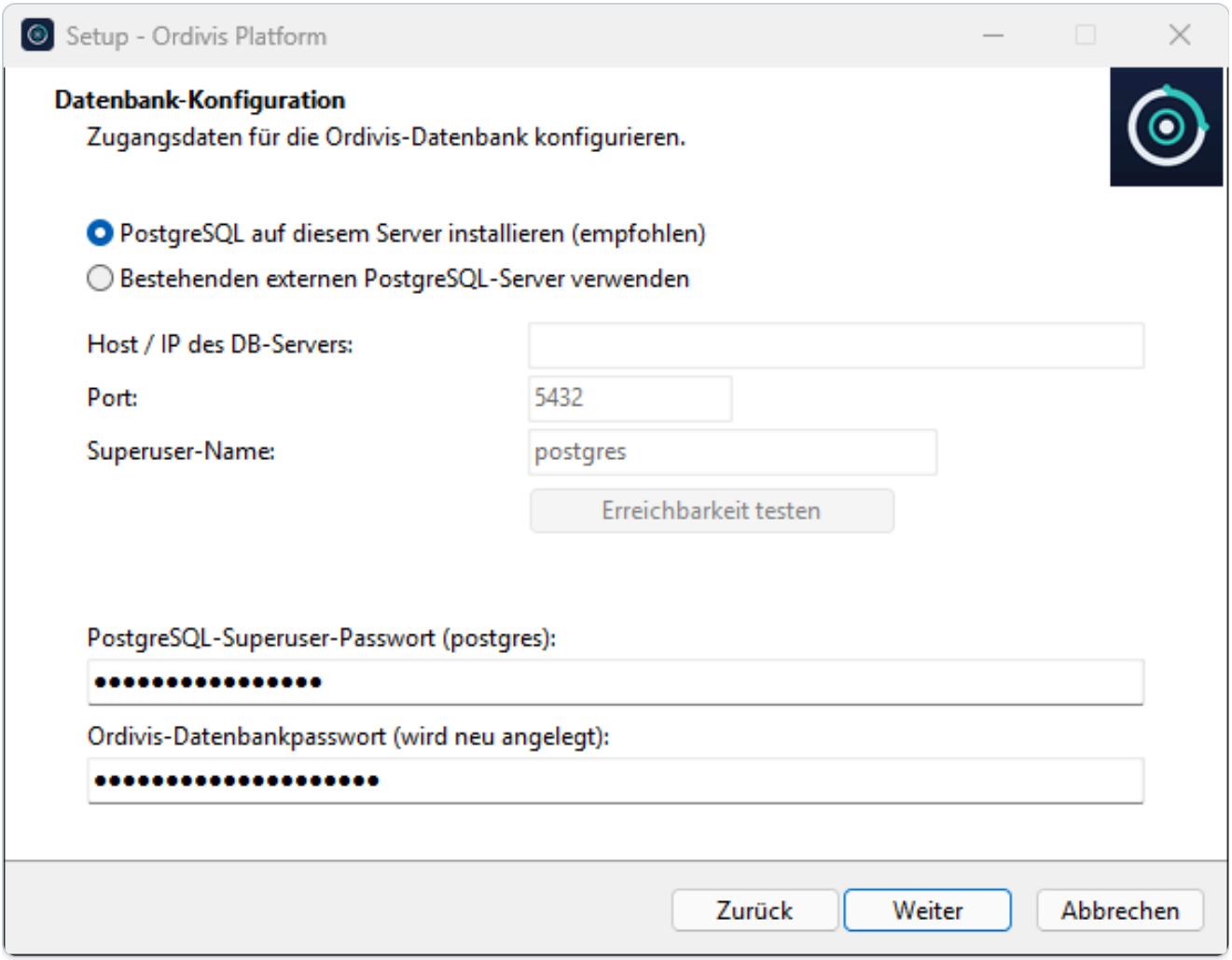

Abb. 5 – **Datenbank**: PostgreSQL lokal installieren (empfohlen) oder einen bestehenden externen Server verwenden. Die Passwörter sind vorbelegt.

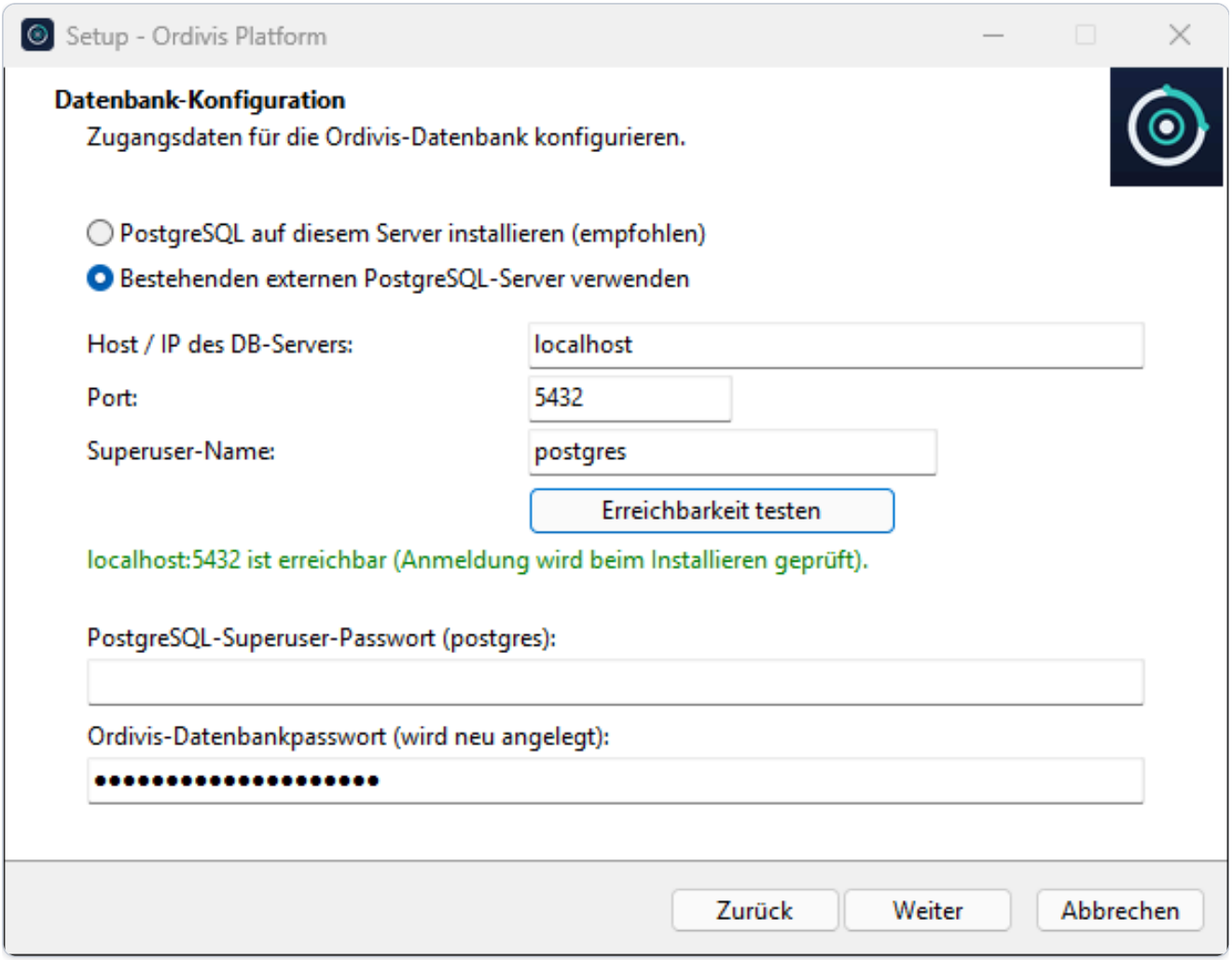

Abb. 6 – Dieselbe Seite mit **bestehendem externem Server**: Host, Port und Superuser-Name werden eingabefähig. **Erreichbarkeit testen** prüft nur, ob der Server antwortet – die Anmeldung selbst wird erst beim Installieren geprüft, und genau das sagt die Meldung auch.

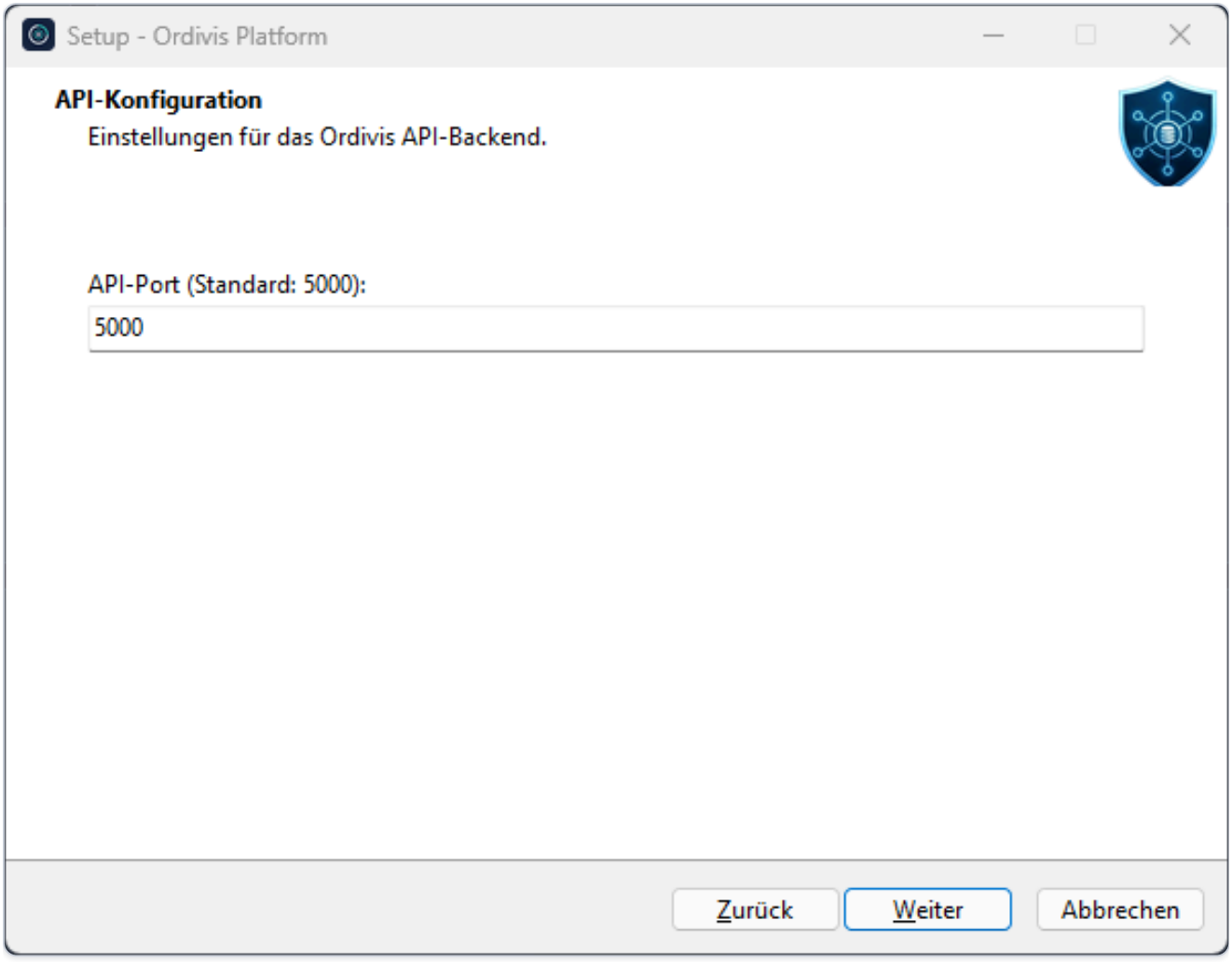

Abb. 7 – **API-Konfiguration**: Port des API-Backends (Standard `5000`). In der Regel unverändert lassen.

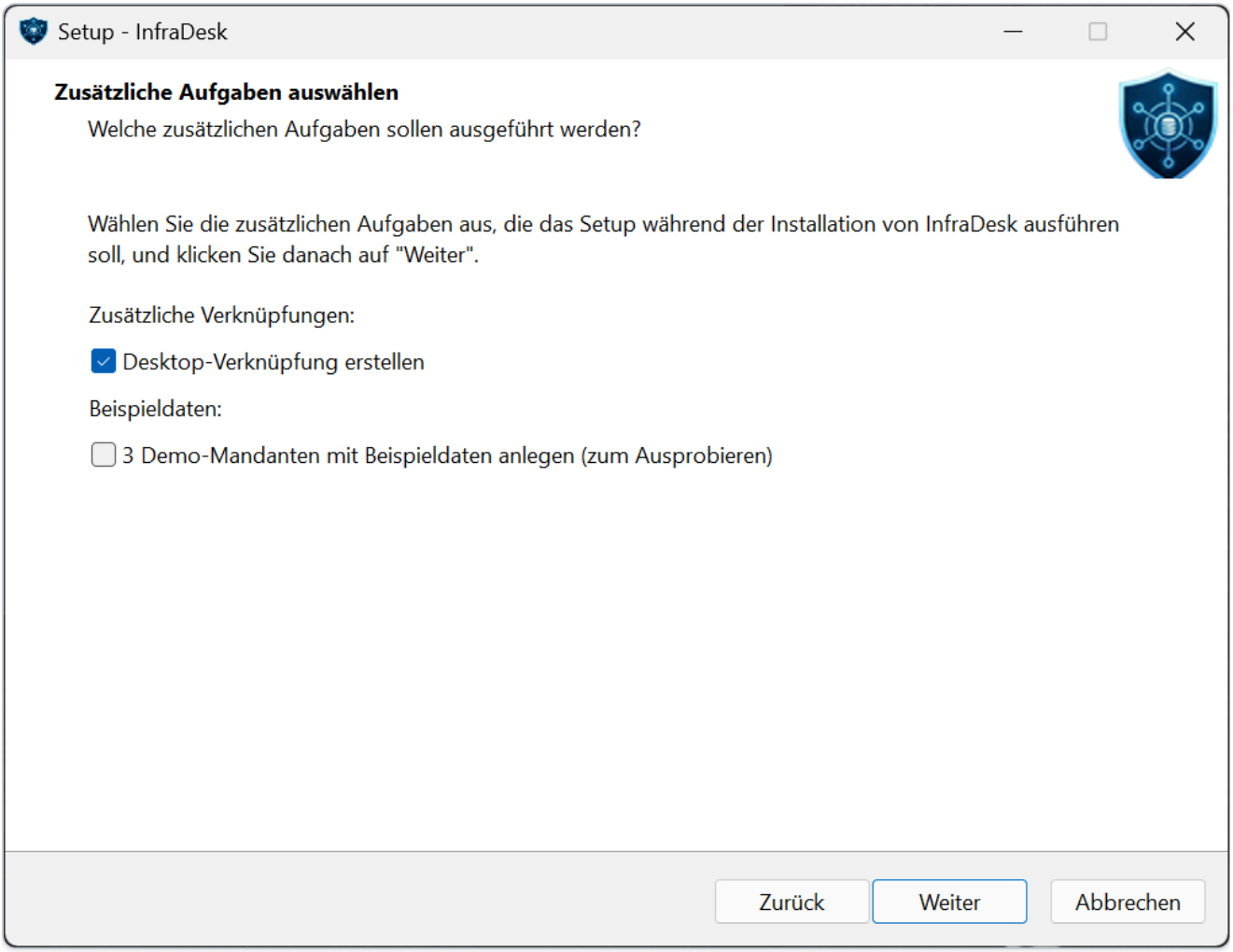

Abb. 8 – **Zusätzliche Aufgaben**: Desktop-Verknüpfung sowie die Option **„3 Demo-Mandanten mit Beispieldaten anlegen"**. Sie füllt drei Vorführ-Mandanten (kleines Unternehmen, Konzern, Kommunalverwaltung) mit anonymen Beispieldaten zum Ausprobieren. **Für Produktivinstallationen deaktiviert lassen.**

Erststart trotz Demo-Mandanten: Die Demo-Mandanten zählen **nicht** als eingerichtetes System. Der Client startet daher auch bei installierten Demo-Mandanten den **Erststart-Assistenten** für den ersten eigenen Mandanten samt **SuperAdmin** (Schritt 6, Erststart/Onboarding). So entsteht immer ein produktiver Mandant mit einem echten SuperAdmin-Konto – unabhängig von den Vorführdaten. Wer die Demo-Mandanten nicht benötigt, lässt die Option ab.

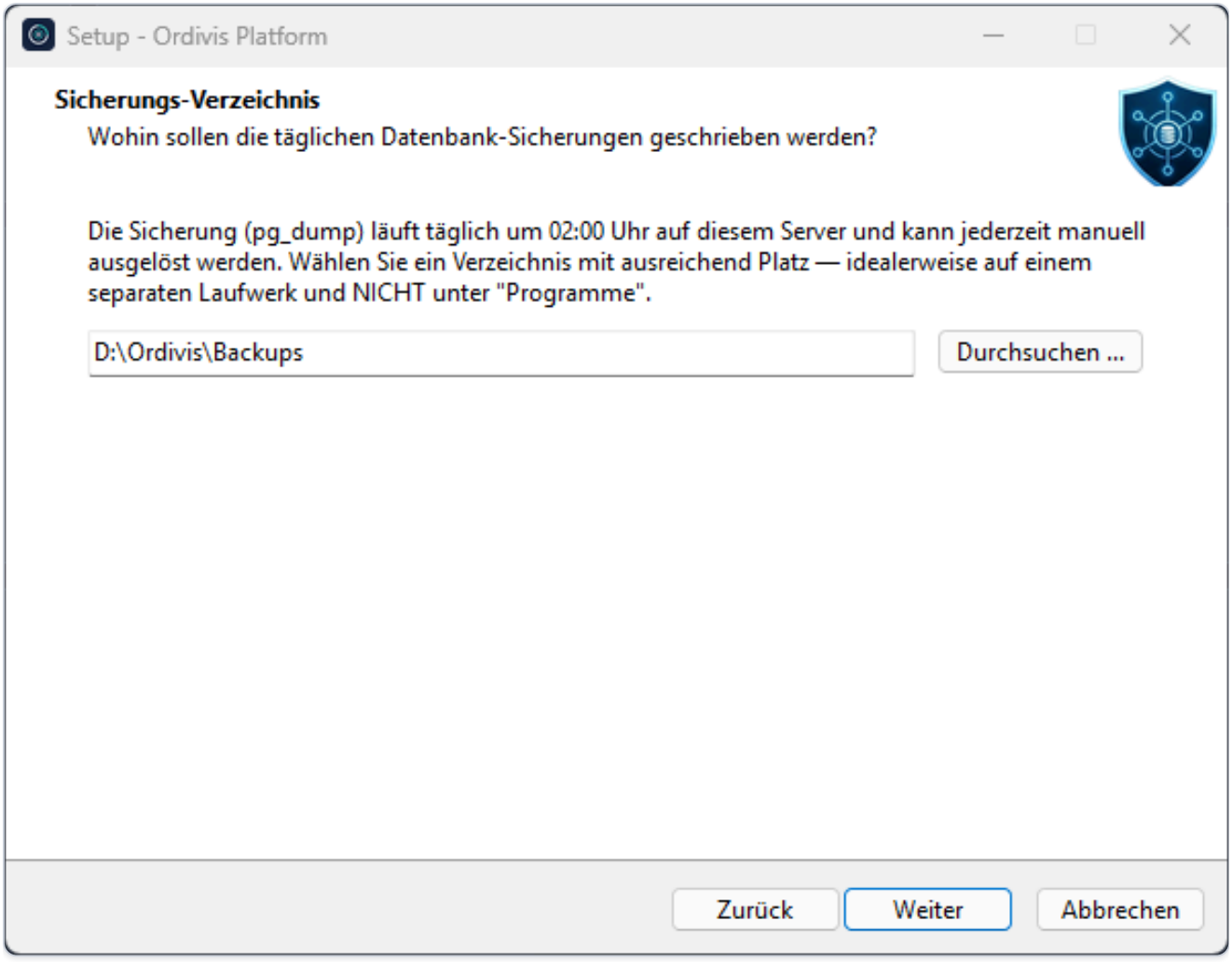

Abb. 9 – **Sicherungs-Verzeichnis** für die tägliche Datenbank-Sicherung (Standard `D:\Ordivis Platform\Backups`, nicht unter „Programme"). Siehe Abschnitt 6 und 4.1.

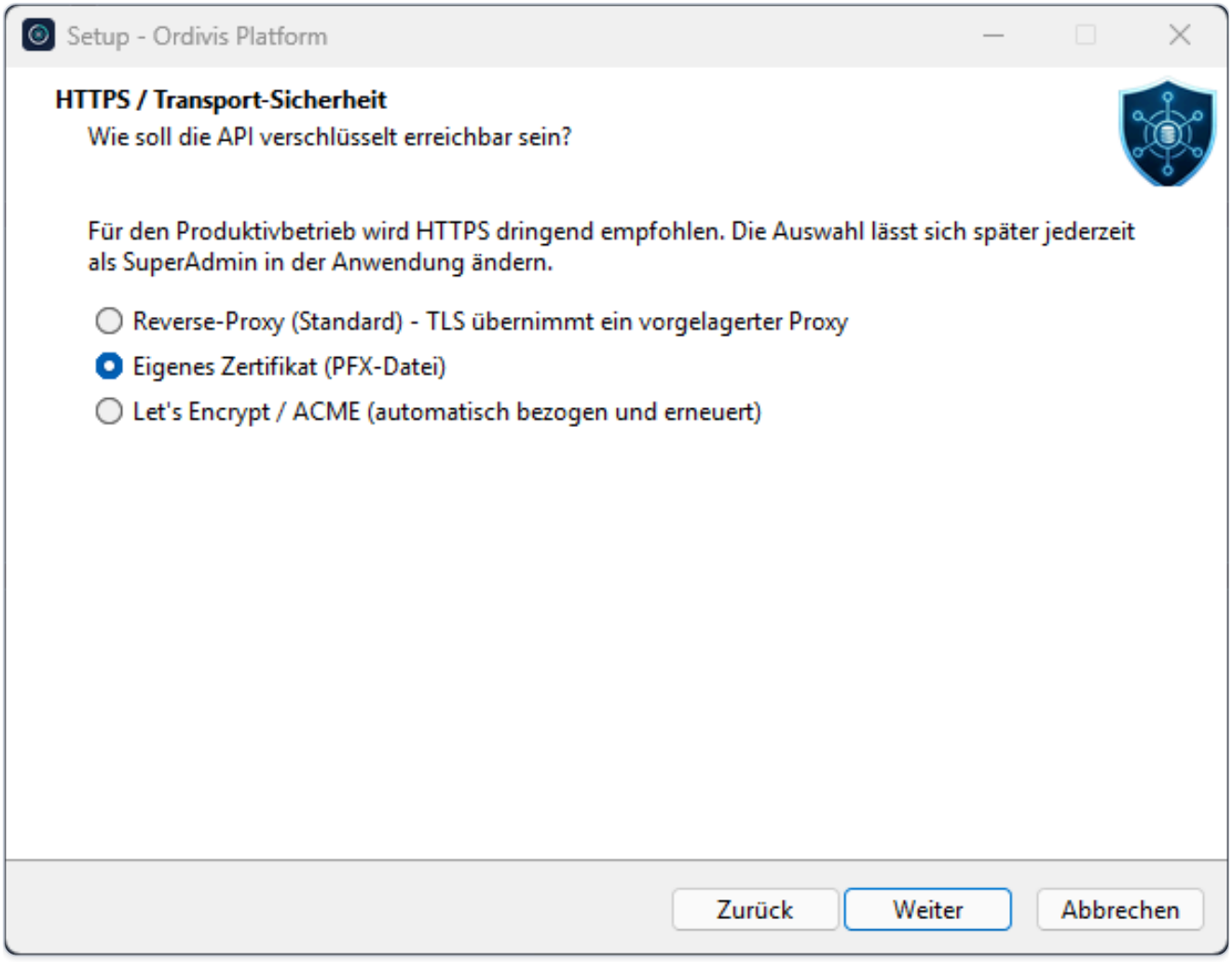

Abb. 10 – **HTTPS / Transport-Sicherheit**: Reverse-Proxy (Vorgabe), eigenes Zertifikat als PFX-Datei oder Let's Encrypt. Die Auswahl lässt sich später jederzeit als SuperAdmin ändern – siehe Abschnitt 7.1.

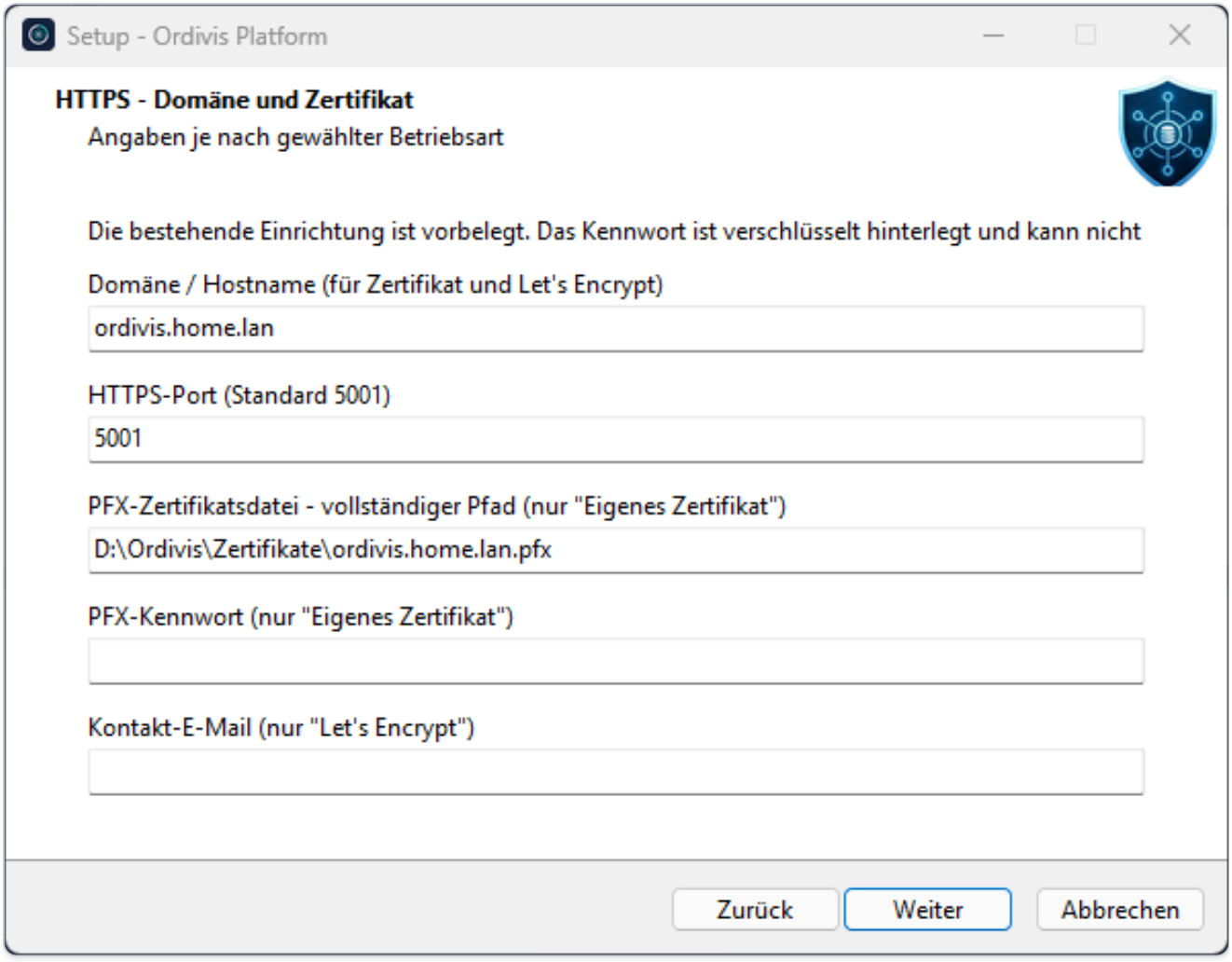

Abb. 11 – **HTTPS – Domäne und Zertifikat**. Auszufüllen sind nur die Felder der gewählten Betriebsart; bei *Reverse-Proxy* wird diese Seite übersprungen. Eine bestehende Einrichtung ist vorbelegt, sodass unverändertes Durchklicken sie unverändert lässt.

Das PFX-Kennwort einer bestehenden Einrichtung wird nicht angezeigt – anders als Domäne, Port und Zertifikatspfad, die vorbelegt erscheinen. Es liegt **DPAPI-geschützt** in der Datei und ist für den Installer, der unter einem anderen Konto läuft, nicht lesbar. **Ein leeres Feld heißt hier „unverändert lassen"**: Der gespeicherte Wert wird übernommen. Nur wer etwas einträgt, ersetzt ihn. Der Assistent sagt das an dieser Stelle auch selbst.

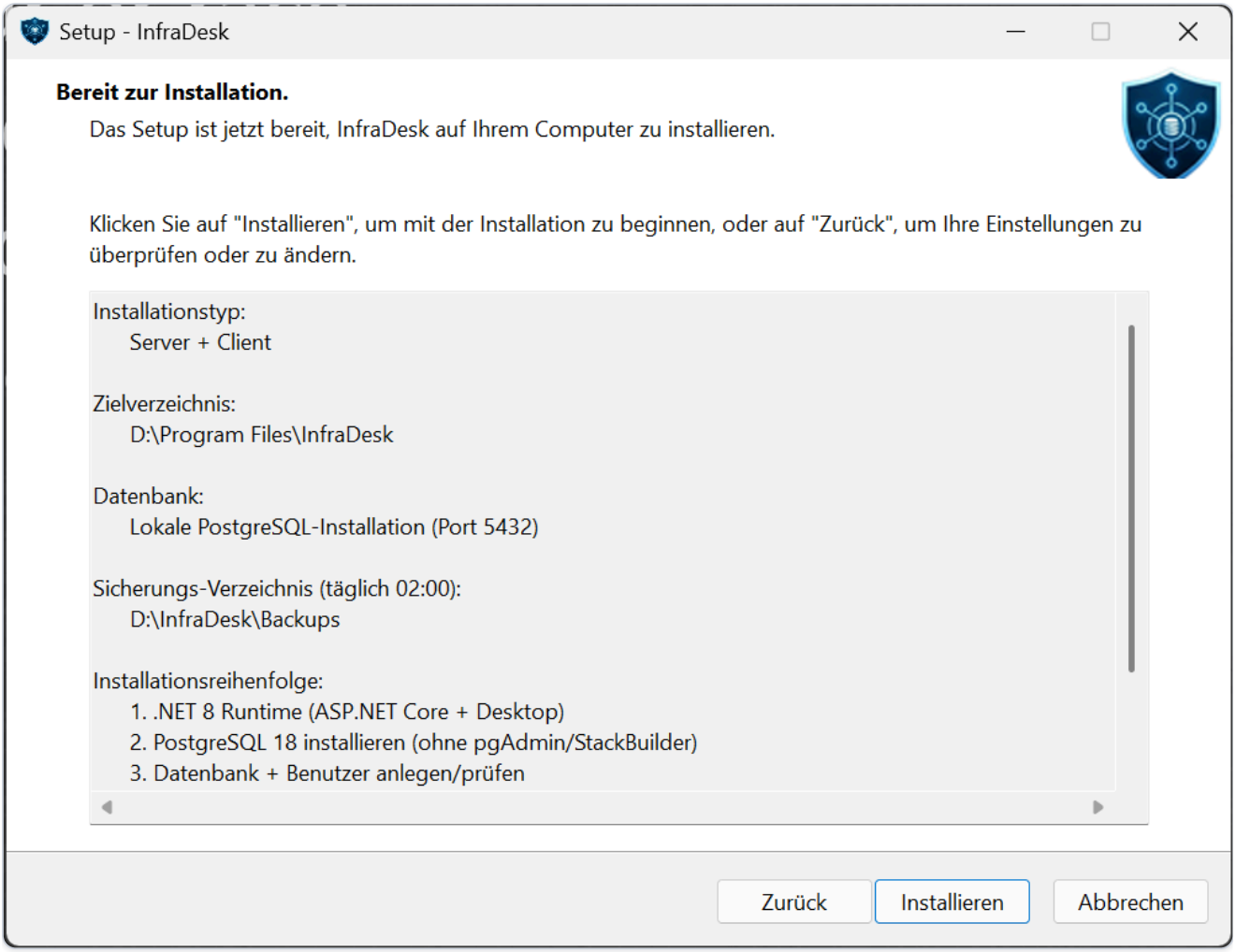

Abb. 12 – **Bereit zur Installation**: Zusammenfassung mit Zielverzeichnis, Datenbank, Sicherungs-Verzeichnis und Installationsreihenfolge.

Nach dem Klick auf **Installieren** läuft die Einrichtung in fünf sichtbaren Schritten ab: (1) .NET-Runtime, (2) PostgreSQL-Datenbankserver, (3) Datenbank und Benutzer anlegen, (4) Windows-Dienste registrieren, (5) Dienst starten und erste Datenbank-Migration. Die beiden längsten Schritte melden sich mit einem ausdrücklichen Hinweis.

Die Abbildungen 13 bis 16 zeigen den Fortschritt und den Abschluss.

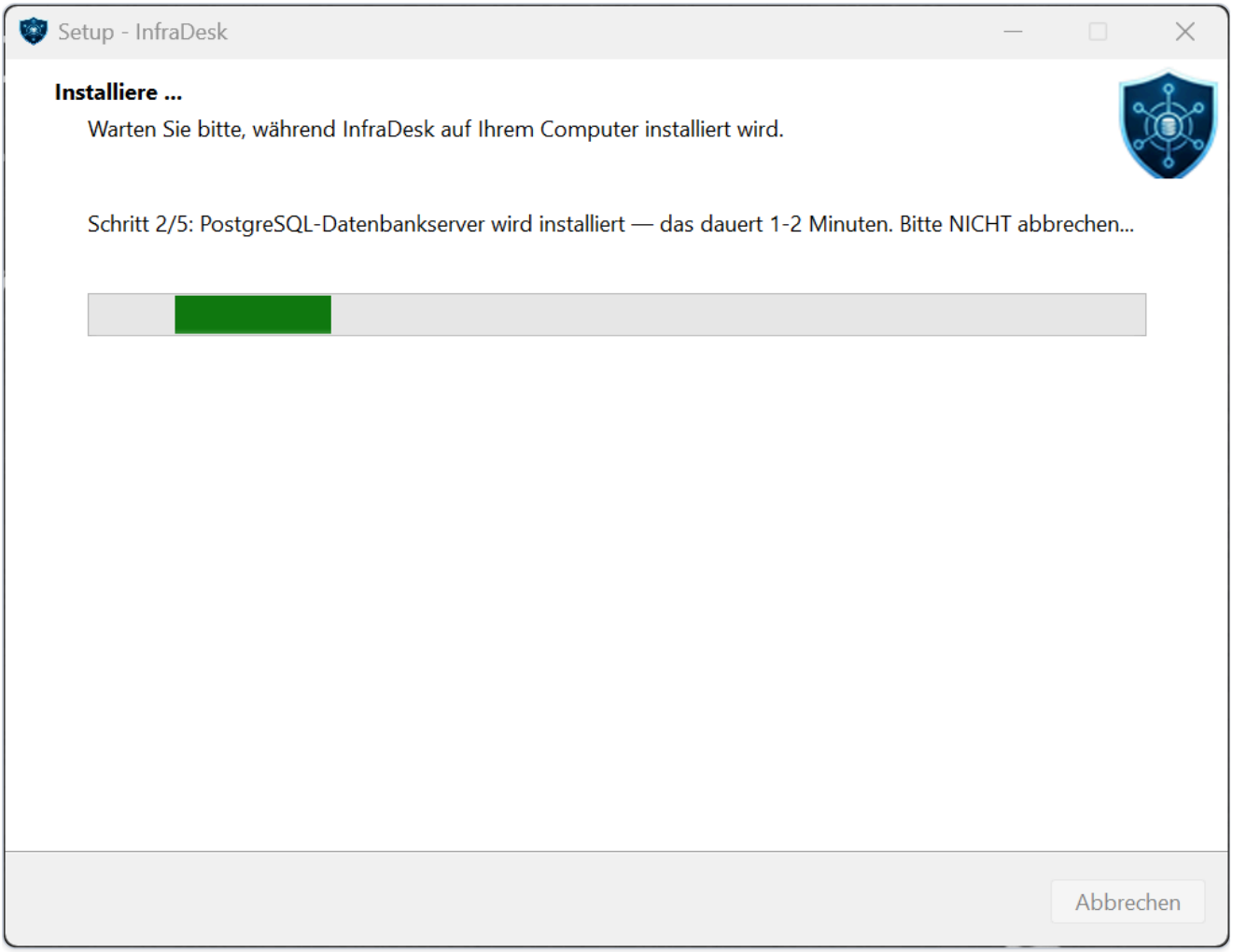

Abb. 13 – **Schritt 2/5**: Der PostgreSQL-Datenbankserver wird installiert (1–2 Minuten). **Bitte nicht abbrechen** – dieser Schritt läuft ohne sichtbares PostgreSQL-Fenster im Hintergrund.

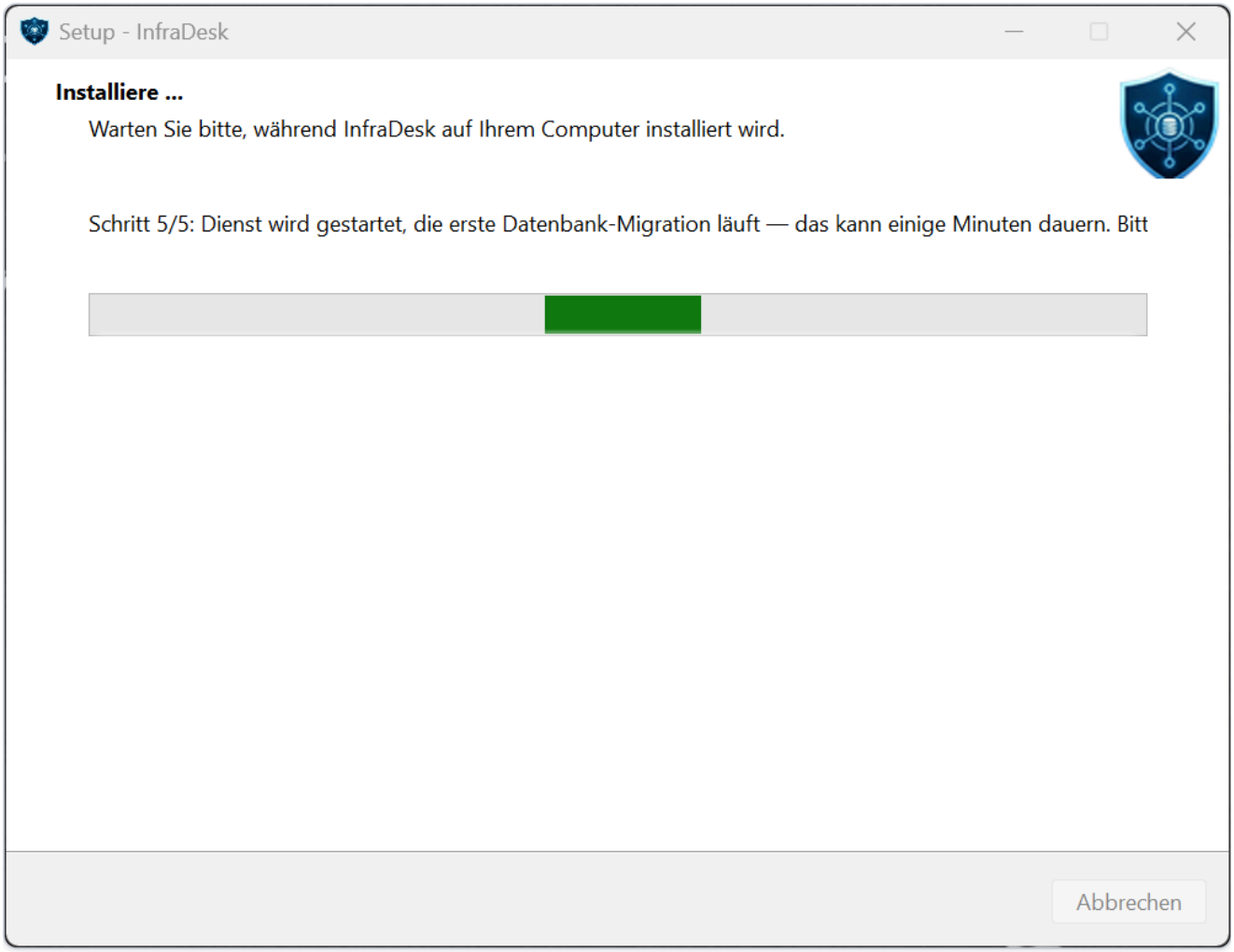

Abb. 14 – **Schritt 5/5**: Die Windows-Dienste starten und die **erste Datenbank-Migration** läuft – das kann einige Minuten dauern. Bitte nicht abbrechen.

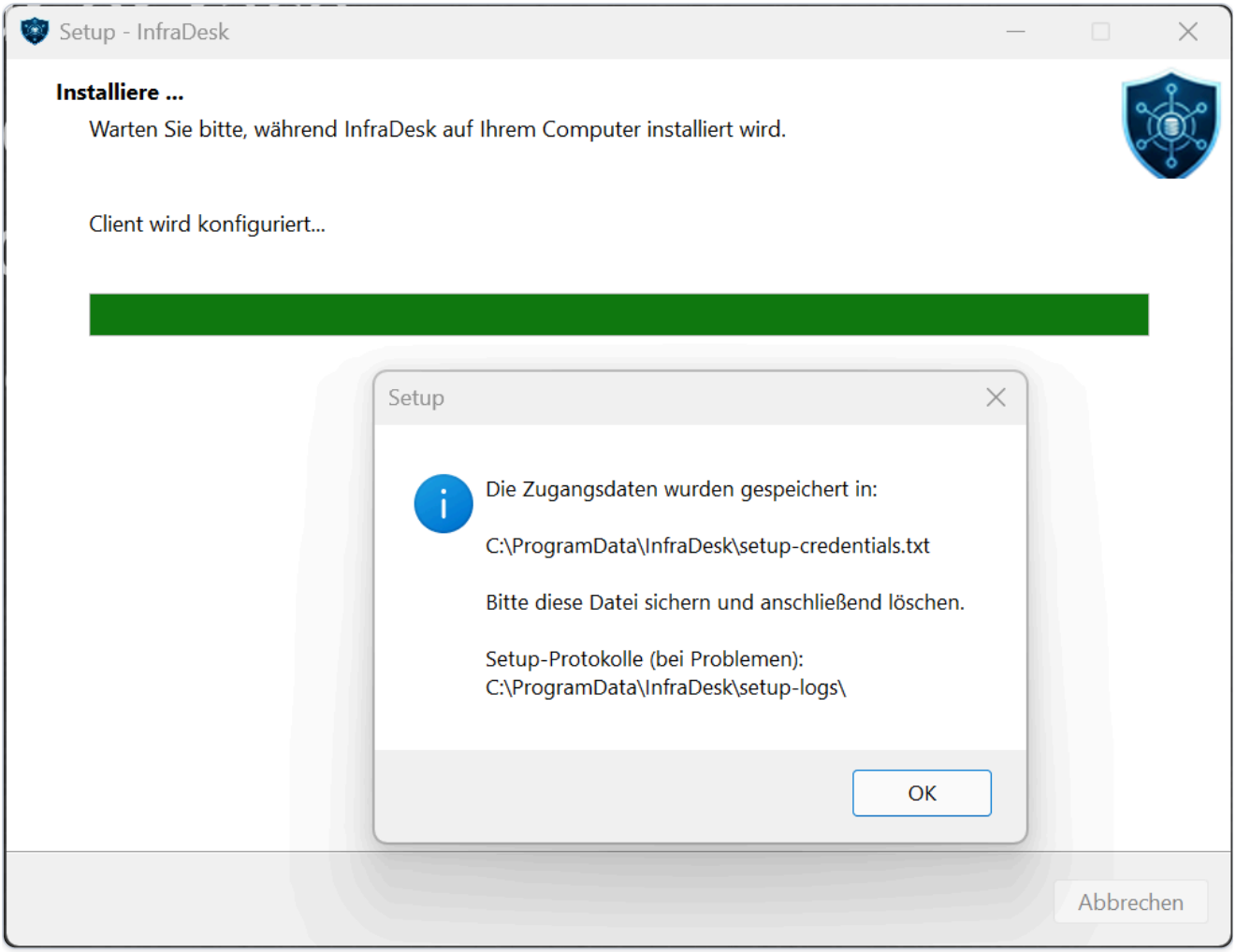

Abb. 15 – Zum Abschluss werden die **Zugangsdaten** in `C:\ProgramData\InfraDesk\setup-credentials.txt` gespeichert. Diese Datei sichern und anschließend löschen.

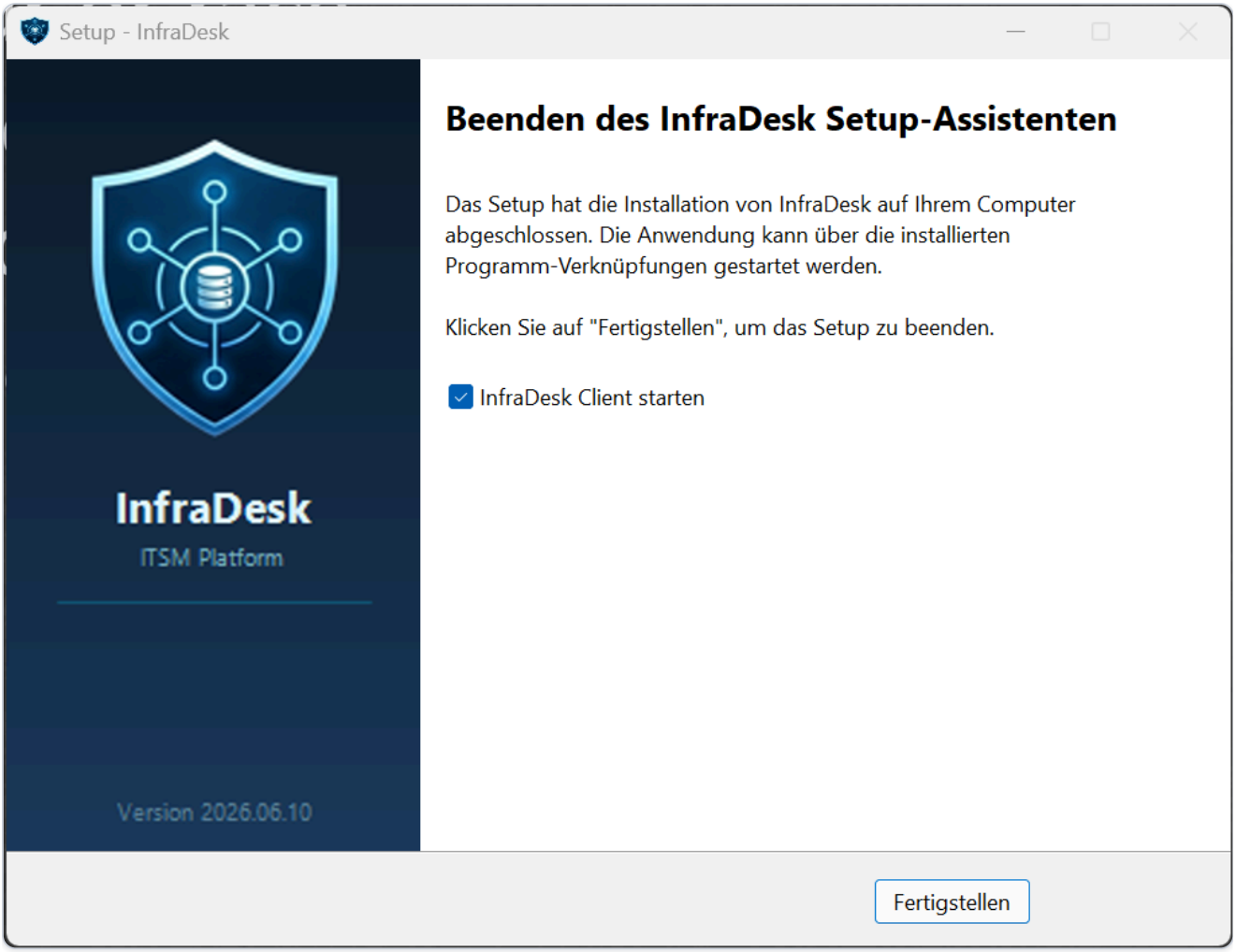

Abb. 16 – Fertig. Beim ersten Start des Clients (oder Aufruf der API-URL) beginnt die geführte Ersteinrichtung (Haupt-Mandant, Zentrale, erster Administrator – siehe ID-40).

2.3 PostgreSQL-Tuning bei der Installation

Bringt der Installer PostgreSQL **selbst** mit, stellt er den Server anschließend automatisch auf den Betrieb ein. Die Werte werden dabei **aus der Zielmaschine berechnet** – Arbeitsspeicher, Prozessorkerne und freier Platz am Datenverzeichnis –, nicht aus einer festen Liste übernommen: Ein Server mit 64 GB und eine VM mit 8 GB brauchen verschiedene Werte.

Gesetzt werden `shared_buffers`, `effective_cache_size`, `work_mem`, `maintenance_work_mem`, `max_connections`, `max_wal_size` / `min_wal_size` sowie `idle_in_transaction_session_timeout`; zusätzlich wird `pg_stat_statements` aktiviert (Grundlage für die Abfrage-Analyse). Geschrieben wird per `ALTER SYSTEM` nach `postgresql.auto.conf` – die von Ihnen gepflegte `postgresql.conf` bleibt unberührt, und jeder Wert lässt sich mit `ALTER SYSTEM RESET <name>` einzeln zurücknehmen.

> **Eine mitgebrachte PostgreSQL-Instanz wird nicht angefasst.** Läuft auf dem Server bereits ein PostgreSQL, oder haben Sie im Assistenten eine **externe Datenbank** gewählt, gehört deren Konfiguration Ihnen: Der Installer überspringt das Tuning dann vollständig und protokolliert das.

Das Protokoll des Laufs steht in `%ProgramData%\InfraDesk\setup-logs\tuning.log` – inklusive der berechneten Werte und der Werte, die der Server nach dem Neustart tatsächlich meldet.

Nachträglich lässt sich das Tuning jederzeit erneut anstoßen (das Skript ist idempotent):

```
# -DryRun rechnet nur und zeigt die Werte, ohne etwas zu schreiben
powershell -File tune-postgresql.ps1 -PgSuperPassword "<Kennwort>" -DryRun
```

> **Tipp:** Eine ausführliche Beschreibung jeder Einstellung samt **empfohlener Werte je Kundengröße** (Lizenzstufe) sowie der Client-Verbindungspools (`Database:MaxPoolSizePerTenant`) steht in ID-51.

3 Konfiguration (appsettings)

- **Connection-String** zur PostgreSQL-Datenbank.
- **Notifications:Smtp** für den E-Mail-Versand (interner Exchange, STARTTLS, FQDN, interne Root-CA).
- **CredentialVault:EncryptionKey** – 32-Byte Base64, **nie** in `appsettings.json`, sondern als Secret/Umgebungsvariable.
- **Identity:Jwt:Secret** – Signaturschlüssel der Anmelde-Token, **mindestens 32 Zeichen**. Ebenfalls **nie** in `appsettings.json`, sondern als Umgebungsvariable (siehe 3.1).
- Beim Kopieren von Updates `appsettings` **ausschließen**, um die Konfiguration zu erhalten.

3.1 Signaturschlüssel der Anmelde-Token (Pflicht)

> **Der Dienst startet seit Version 2026.07.14.1845 nicht mehr, wenn kein eigener Signaturschlüssel gesetzt ist.** Das ist beabsichtigt.

Die ausgelieferte `appsettings.json` enthält für `Identity:Jwt:Secret` lediglich einen **Platzhalter**. Bliebe dieser Wert stehen, liefe die Anwendung mit einem Schlüssel, der in **jeder** Installation identisch und damit öffentlich bekannt ist – jeder könnte sich damit ein gültiges Zugangstoken ausstellen, mit beliebiger Identität und beliebigen Rechten. Ein solcher Zustand darf nicht unbemerkt weiterlaufen; die Anwendung verweigert deshalb den Start und nennt im Ereignisprotokoll den Grund.

Für Installationen über `setup.ps1` ist nichts zu tun: Das Setup erzeugt bei der Installation automatisch einen zufälligen 48-Byte-Schlüssel und hinterlegt ihn als Maschinen-Umgebungsvariable `Identity__Jwt__Secret`. Diese bleibt bei Updates erhalten.

Manuell setzen (nur nötig, wenn die Installation nicht über `setup.ps1` erfolgte):

```
# 48 Byte Zufall → Base64 (64 Zeichen)
$bytes = New-Object byte[] 48
[System.Security.Cryptography.RandomNumberGenerator]::Create().GetBytes($bytes)
$secret = [Convert]::ToBase64String($bytes)

[System.Environment]::SetEnvironmentVariable("Identity__Jwt__Secret", $secret, "Machine")
Restart-Service InfraDesk.Api
```

> **Hinweis:** Wird der Schlüssel geändert, werden alle bestehenden Anmeldungen ungültig – die Benutzer melden sich einmal neu an.

3.2 Betrieb hinter einem Reverse-Proxy (Pflichtangabe seit 2026.08)

Steht ein Reverse-Proxy (IIS, nginx, HAProxy) vor der API, sieht diese ohne weitere Angabe **für jeden Aufruf dieselbe Adresse** – nämlich die des Proxys. Alle Schutzmechanismen, die je Adresse zählen (Anmeldeversuche, Aufrufgrenzen), teilen sich dann **eine gemeinsame Zählung für das ganze Haus**: Ein einzelner Nutzer kann damit die Anmeldung für alle anderen sperren.

> 🔑 Zugriff auf `appsettings.Production.json` des Servers und das Recht, den Dienst `InfraDesk.API` neu zu starten — nicht in Client oder Portal einstellbar.

🔒 **Pflichtreihenfolge** — die Angabe wirkt erst nach einem Neustart des Dienstes. Bis dahin zählen alle Aufrufe weiter gegen dieselbe Adresse.

1. Ermitteln Sie die Adressen aller Proxys, die vor der API stehen.
2. Tragen Sie sie in `appsettings.Production.json` ein (Beispiel unten).
3. Starten Sie den Dienst `InfraDesk.API` neu.
4. Prüfen Sie im Protokoll, dass Anmeldeversuche wieder je Ursprungsadresse gezählt werden.

✅ **Ergebnis:** Ein einzelner Nutzer kann die Anmeldung nicht mehr für das ganze Haus sperren.

```
"Hosting": {
  "TrustedProxies": [ "10.0.0.5", "10.0.0.6" ]
}
```

> **Die Liste ist bewusst leer vorbelegt.** Solange sie leer ist, werden `X-Forwarded-For`-Angaben **ignoriert**. Das ist richtig so: Würde die API sie ungeprüft glauben, könnte jeder Aufrufer eine beliebige Herkunft behaupten und die Aufrufgrenzen umgehen. Eintragen also **nur** die Adressen, von denen wirklich Ihr Proxy kommt – kein `*`, kein ganzes Netz.

Läuft die API **ohne** Proxy direkt am Netz, bleibt die Liste leer.

3.3 Wetter und amtliche Unwetterwarnungen einrichten

Wetterlage, Dreitagesvorschau, die **amtlichen Unwetterwarnungen** des Deutschen Wetterdienstes, die amtliche Warnkarte und der Wald- und Graslandfeuerindex erscheinen im **Web-Portal, im WinUI-Client und auf dem Wallboard**.

> **Achtung – seit 2026.08 stehen die Einstellungen woanders.** Der Abschnitt `Weather` gehört jetzt in die `appsettings.json` der **API** (`InfraDesk.App`), früher stand er beim **Portaldienst**. Nur die API fragt noch nach draußen; die Anzeigeprogramme holen alles von ihr. Wer `Weather:*` beim Portal gepflegt hatte, trägt die Werte um – dort werden sie **nicht mehr gelesen**, ohne Fehlermeldung.

```
"Weather": {
  "Enabled": true,
  "StationIdOverride": "",
  "LocationOverride": "",
  "CacheMinutes": 15
}
```

Wert	Bedeutung
`Enabled`	Auf `false` setzen, wenn der Server **keinen Weg ins Internet** hat. Sonst läuft jeder Abruf in einen Zeitablauf.
`StationIdOverride`	**Leer lassen.** Die Station ergibt sich dann aus der **Postleitzahl im Mandantenprofil**. Ein Eintrag hier übersteuert sie – in einer Installation mit mehreren Mandanten für **alle** zugleich. Gedacht für den Einzelfall, in dem die abgeleitete Station danebenliegt.
`LocationOverride`	Anzeigename zur übersteuerten Station. Der DWD liefert zur Kennung **keinen Klarnamen** mit – bliebe das Feld leer, stünde auf der Seite eine nackte Zahl. Ohne Übersteuerung unbenutzt.
`CacheMinutes`	Wie lange ein Abruf gilt. Der DWD rechnet stündlich; kleinere Werte bringen keine neueren Zahlen, belasten aber eine fremde, kostenlose Schnittstelle.

Der frühere Schlüssel `StationId` wird nicht mehr gelesen. Er trug die ausgelieferte Vorgabe `10384` (Berlin-Tempelhof) und stand damit in *jeder* Installation – genau deshalb sah jeder Mandant das Berliner Wetter. Tragen Sie stattdessen die Postleitzahl im Mandantenprofil ein.

Warnungen sind gemeindescharf, das Wetter ist es nicht. Der DWD warnt nicht für Messstationen, sondern für **Warnzellen** (in aller Regel Gemeinden). Messwerte und Vorschau hängen weiterhin an der nächstgelegenen Station, die amtlichen Warnungen dagegen an Ihrer Gemeinde. Beide Angaben können daher unterschiedliche Ortsnamen tragen – das ist richtig so und keine Fehlkonfiguration.

Netzfreigabe. Benötigt wird ausgehend **HTTPS (443)** vom **API-Server** (nicht vom Portal) zu:

Ziel	Wofür	Einstellbar über
`app-prod-ws.warnwetter.de`	Messwerte und Dreitagesvorschau	fest
`api.brightsky.dev`	amtliche Warnungen, aufgelöst auf Warnzellen	`Weather:AlertsBaseUrl`
`maps.dwd.de`	amtliche Warnkarte (WMS)	`Weather:MapBaseUrl`
`opendata.dwd.de`	Wald- und Graslandfeuerindex	`Weather:OpenDataBaseUrl`

Übertragen werden dabei keine Nutzer-, Mandanten- oder Gerätedaten – nur Stationskennung bzw. Koordinaten des abgefragten Ortes.

Für den Katastrophenschutz: Bright Sky lässt sich selbst betreiben. Der Dienst bereitet die Offenen Daten des DWD als JSON auf und löst Koordinaten zu Warnzellen auf. Er ist quelloffen; wer die Abhängigkeit von einem fremden Dienst nicht tragen will, betreibt ihn im eigenen Netz und trägt die eigene Adresse unter `Weather:AlertsBaseUrl` ein.

Wer wird benachrichtigt. Bei einer **neuen** amtlichen Warnung benachrichtigt die Anwendung alle Konten mit dem Recht `weather.warning.receive` – in aller Regel über die mitgelieferte Rolle **„Wetter-Warnung“**. Sie ist ein reiner Verteiler ohne jede Befugnis und wird **zusätzlich** zur eigentlichen Rolle vergeben. Dieselbe Warnung weckt niemanden zweimal, auch wenn der DWD sie über zwei Tage hinweg wiederholt.

3.4 Windows-Anmeldung im Web-Portal (seit 2026.08)

Im Hausnetz kann sich ein bereits am Windows-Rechner angemeldeter Benutzer **ohne Kennworteingabe** im Portal anmelden. Die Anmeldemaske versucht das beim Aufruf einmal von selbst; scheitert es, steht die gewöhnliche Anmeldung mit Benutzername und Kennwort unverändert bereit.

Warum es zwei Bausteine braucht. Der Browser handelt die Windows-Anmeldung mit dem **Portal** aus, nicht mit der Schnittstelle. Diese Identität lässt sich nicht weiterreichen – dafür wäre eine eingeschränkte Kerberos-Delegierung nötig, also ein Eingriff in Ihren Verzeichnisdienst. Stattdessen prüft das Portal selbst und weist sich bei der Schnittstelle mit einem **gemeinsamen Geheimnis** aus.

Vier Angaben, die zusammen gelten. Fehlt eine, bleibt die Windows-Anmeldung im Portal aus – die Anmeldung mit Kennwort funktioniert weiterhin:

Wo	Einstellung	Bedeutung
Portal	`WindowsSso:Enabled`	Auf `true` setzen.
Portal (Dienstkonto)	–	Der Portaldienst muss unter einem **Domänenkonto** mit passendem SPN laufen, nicht unter `LocalSystem`.
Portal	`WindowsSso:TrustSecret`	Das gemeinsame Geheimnis.
API	`Identity:WindowsSso:PortalTrustSecret`	**Derselbe** Wert wie oben.
API	`Identity:WindowsSso:TrustedOrigins`	Die Adresse des Portals, z. B. `https://portal.firma.local`.

> **Die Liste `TrustedOrigins` ist bewusst leer vorbelegt, und leer bedeutet ausdrücklich „nur vom selben Rechner", nicht „von überall".** Das Geheimnis allein soll nicht genügen: Wer es erbeutet, könnte sonst von jedem Rechner im Netz eine beliebige Identität behaupten. Tragen Sie die Portaladresse ein, sobald Portal und Schnittstelle auf verschiedenen Rechnern laufen.

Das Geheimnis wird verschlüsselt abgelegt. Steht es im Klartext in der `appsettings.json`, bemängelt die Schnittstelle das beim Start im Protokoll. Verschlüsseln Sie es mit dem Maschinenschlüssel; der Wert trägt dann das Präfix `DPAPI:`.

Ohne den Schalter der Client-Anmeldung. Die Windows-Anmeldung des Portals hängt **nicht** an `Identity:WindowsSso:Enabled`. Dieser Schalter steuert das Anmeldeverfahren der Schnittstelle selbst (für den Windows-Client) und kann aus anderen Gründen aus sein, etwa hinter einem Reverse-Proxy. Der Schalter des Portalwegs ist das hinterlegte Geheimnis.

Was nicht geht. Mandantenwahl und zweiter Faktor führen weiterhin über die gewöhnliche Anmeldung – beides gehört dorthin. Und das Notfallkonto ist über die Windows-Anmeldung **nicht** erreichbar; das ist Absicht.

3.5 Sitzungsdauer (seit 2026.08)

Wie lange eine Anmeldung hält, steht an **zwei** Stellen — je einmal für die Schnittstelle und für das Web-Portal. Beide Fristen sind **gleitend**: Jeder Zugriff setzt sie neu, sie laufen also erst nach so langer **Ruhe** ab. Sie sind keine Höchstdauer; ein geöffneter Windows-Client hält seine Sitzung durch seine regelmäßigen Abrufe von allein offen.

Anwendung	Einstellung	Vorgabe
Schnittstelle (Windows-Client, Mobil)	`Identity:Jwt:RefreshTokenExpiryHours`	12 Stunden
Web-Portal	`Session:LifetimeMinutes`	30 Minuten

Der große Unterschied ist Absicht. Am Windows-Client arbeitet eine benannte Person an einem Arbeitsplatz, den sie sperrt; das Portal steht Endanwendern offen und läuft auch an gemeinsam genutzten Rechnern, etwa im Bürgerbüro oder in der Werkstatt. Eine halbe Stunde Ruhe ist dort die passendere Grenze. Wer längere Portal-Sitzungen braucht, trägt einen höheren Wert ein — `480` wären die früheren acht Stunden.

> ⚠️ **Wo die Werte der Schnittstelle einzutragen sind.** Auf einem installierten Server gelten die **Maschinen-Umgebungsvariablen** `Identity__Jwt__RefreshTokenExpiryHours` und `Identity__Jwt__AccessTokenExpiryMinutes`; der Installer setzt sie. Sie haben Vorrang vor `appsettings.Production.json` — und sind zugleich der einzige Weg, der eine **bestehende** Installation erreicht, denn Updates kopieren `appsettings*.json` bewusst nicht mit (Abschnitt 3, letzter Punkt).

Der Wert des Portals steht in dessen `appsettings.json`; sie überlebt Updates aus demselben Grund. Nach einer Änderung den jeweiligen Dienst neu starten (`InfraDesk.API` bzw. `InfraDesk.Portal`).

> ⚠️ `Session:LifetimeMinutes` **auf 0 oder darunter ist keine Laufzeit, sondern ein Tippfehler** — er würde jeden Anwender sofort abmelden. Das Portal nimmt in diesem Fall die Vorgabe und schreibt eine Warnung ins Protokoll.

Das kurzlebige Zugriffstoken (`Identity:Jwt:AccessTokenExpiryMinutes`, 15 Minuten) ist **nicht** die Sitzungsdauer: Es wird im Hintergrund erneuert, ohne dass jemand es merkt. Es hochzusetzen verlängert nur die Zeit, bis entzogene Rechte oder eine gesperrte Kennung greifen — für die Anwender ändert es nichts.

4 Datenbank-Migrationen

Migrationen werden mit der Auslieferung bereitgestellt und beim Update angewendet.

> **Achtung:** Neue Tabellen müssen der App-Rolle gehören. Wird eine Migration versehentlich als `postgres` gegen die echte Datenbank getestet, gehören die Tabellen `postgres`, und die als `infradesk_app` laufende API erhält **42501 / DbUpdateException**. Abhilfe: `ALTER TABLE … OWNER TO infradesk_app`.

4.1 Mandanten-Provisionierung (eigene Datenbank je Mandant)

Legt ein SuperAdmin über **Einstellungen → Mandanten** einen neuen Mandanten an, erzeugt die Anwendung dafür eine **eigene Datenbank** (`infradesk_<slug>`). Dazu muss die verwendete Datenbank-Rolle das Recht `CREATEDB` besitzen.

Bei einer **Neuinstallation** wird das bereits eingerichtet: Setup/Installer legt die App-Rolle `infradesk_app` mit `CREATEDB` an – die Mandanten-Provisionierung funktioniert also **direkt**.

Nur bei **älteren Bestandsinstallationen** kann das Recht fehlen; dann bricht das Anlegen mit **„42501: keine Berechtigung, um Datenbank zu erzeugen“** ab. Abhilfe (als PostgreSQL-Superuser):

```
ALTER ROLE infradesk_app CREATEDB;
```

> **Härtungs-Alternative (KRITIS/BSI):** Statt der App-Rolle `CREATEDB` zu geben, kann in `appsettings.json` unter `ConnectionStrings` ein Eintrag `OrdivisMaintenance` mit einer eigenen, nur zum Anlegen von Datenbanken berechtigten Rolle hinterlegt werden. Das Provisioning nutzt diese Verbindung bevorzugt; die App-Rolle bleibt dann ohne `CREATEDB` (Least Privilege).

Das **Sicherungs-Verzeichnis** wird im Setup abgefragt (Standard nicht unter `C:\ProgramData`, sondern ein dediziertes Datenverzeichnis, z. B. `D:\Ordivis Platform\Backups`) und sowohl an die geplante Aufgabe (`backup.ps1 -BackupDir`) als auch an die Anwendung (`Backup:Directory`) übergeben.

5 In-Place-Update & Versionierung

- Die App-Version folgt dem Schema **JJJJ.MM.TT.HHMM**.
- Update: Dienst stoppen → Dateien kopieren (ohne `appsettings`) → Migrationen → Dienst starten.

> **Achtung:** Der erste `Start-Service` nach dem Kopieren der API scheitert häufig am SCM-Timeout/Kaltstart – **einfach erneut starten**.

> **Achtung (IPAM/Deploy-Falle):** Stoppt der Dienst nicht innerhalb von 60 s, bleiben geladene DLLs gesperrt und werden nicht überschrieben (Symptom: 404 statt 401). Dann den Prozess **elevated per** `taskkill` beenden und erneut deployen.

> **Hinweis (Client/WinUI):** Zum Bauen des Clients `/t:Build` verwenden (nicht `/t:Publish`) – ein Publish entfernt die PRI-Ressourcen, wodurch der Client nach dem Splash abstürzt (XamlParseException).

5.1 Delta-Update (signiert, client-gesteuert)

Ergänzend zum manuellen In-Place-Update gibt es das **Delta-Update-Verfahren**: Statt das gesamte Setup zu übertragen, wird je Version ein **signiertes Release-Manifest** ausgeliefert, und das Ziel lädt nur die **abweichenden** Dateien.

- **Release-Manifest:** je Produktversion ein mit **ECDSA (ES256)** signiertes `manifest.json` mit der Datei-Liste (Pfad, SHA-256, Größe) pro Komponente (API, Portal, Client, Wallboard und seit 2026.08.12.1127 der Aktualisierungsdienst des Arbeitsplatzes, siehe 5.1.2). Der Hersteller erzeugt es mit dem Werkzeug `tools/InfraDesk.UpdateManifestGenerator` aus den publizierten Dist-Ordnern; der private Signaturschlüssel bleibt ausschließlich beim Hersteller, der **öffentliche** Schlüssel ist im Produkt eingebettet. Ein Manifest ohne gültige Signatur wird **nie** angewandt.
- **Update-Quelle:** `https://…/updates/<version>/` mit `manifest.json` und **inhaltsadressierter Blob-Ablage** (`<component>/<hash>`) – identische Dateien über Versionen hinweg teilen sich einen Blob.
- **Delta am Ziel:** Das Produkt vergleicht die installierten Datei-Hashes mit dem Manifest und lädt nur die fehlenden/geänderten Blobs (jeweils gegen ihren Hash verifiziert) in ein **Staging-Verzeichnis** (`%ProgramData%\InfraDesk\update-staging\<version>`) – **nicht-destruktiv**, die laufende Installation bleibt unangetastet.

- **Anstoßen (SystemAdmin):** `GET /api/v1/system/update/delta-status` zeigt den signatur-geprüften Umfang (MB/Dateien); der Client zeigt ihn unter *Einstellungen ▸ Updates* hinter „Nach Updates suchen". `POST /api/v1/system/update/stage/<version>` lädt die Blobs. `activation.manage` zum Lesen, `system.config.manage` zum Anstoßen.
- **Anwenden & Rollback (entkoppelter Updater):** Das eigentliche Übernehmen aus dem Staging läuft seit Version **2026.07.15.1738 nicht mehr in der laufenden API**, sondern in einem eigenständigen `InfraDesk.Updater` (Bootstrapper), der den Stopp von API und Client übersteht. Er wendet **alle lokalen Komponenten** an – je Komponente: Dienst **sauber stoppen** (`sc stop` → auf STOPPED warten → nur zur Not `taskkill /F` gegen DLL-Locks) → **Snapshot** anlegen → Dateien übernehmen (mit Copy-Retry) → **starten** → **Health**. Schlägt ein Schritt fehl, wird der Snapshot **zurückgespielt** (Vorversion). Vor dem gesamten Lauf wird ein **DB-Backup** angestoßen. Das geprüfte Ergebnis je Komponente landet in `%ProgramData%\InfraDesk\update-status.json` (der Client zeigt es beim nächsten Start).
- **Log:** je Host `%ProgramData%\InfraDesk\update-logs\` plus zentrales Update-Log (DB, Schema `system`) und Audit-Ereignis `system.update.performed`.

> **Hinweis (Reihenfolge & verteilte Hosts):** Der Orchestrator im Backend stellt einen Gesamtplan über alle betroffenen Komponenten zusammen und übergibt ihn dem entkoppelten Updater. Reihenfolge: **Dienste zuerst** (Portal, dann API – die API zuletzt unter den Diensten, damit sie ihre HTTP-Antwort noch senden kann), der **Client ganz zuletzt** (er wird erst nach gesunder API neu gestartet, und nur, wenn er lief). Liegen API und Portal auf **getrennten Hosts**, wird die jeweils fremde Komponente über deren lokalen **Update-Agent** aktualisiert.

5.1.1 ARBEITSPLATZ-CLIENTS (GETRENNTE INSTALLATION)

Der Orchestrator plant ausschließlich Komponenten, deren Verzeichnis auf dem **Server-Host** liegt. Ein Client, der auf einem eigenen Arbeitsplatz-PC installiert ist, wird davon **nicht erfasst** – auch dann nicht, wenn das Update von genau diesem Client ausgelöst wurde. Dafür sorgt seit **2026.07.22.1615** ein eigener Dienst auf dem Arbeitsplatz.

Aktualisierungsdienst `InfraDesk.ClientUpdate`

Bei einer **reinen Client-Installation** richtet das Setup diesen Windows-Dienst ein (verzögerter Automatikstart, Konto `LocalSystem`). Auf einem Host, auf dem auch die API läuft, wird er **nicht** registriert – dort erledigt der Orchestrator die Client-Kopie bereits.

- **Zeitplan:** Der Dienst fragt standardmäßig alle **4 Stunden** beim eigenen Server nach der Zielversion. Er ist von außen nicht erreichbar (kein eingehender Port, keine Firewall-Regel nötig).
- **Zielversion** ist die **laufende Version des eigenen Servers**, nicht die neueste auf der Produkt-Website. So bleiben Arbeitsplatz und Backend im Gleichschritt; angehoben wird der Arbeitsplatz erst, wenn der Server selbst aktualisiert wurde.
- **Erst laden, dann fragen:** Manifest (**ES256**) und Blobs (**SHA-256** je Blob) werden vollständig geprüft heruntergeladen, bevor irgendjemand gefragt wird. Die Installation bleibt dabei unberührt.
- **Läuft der Client nicht**, wird sofort und ohne Rückfrage aktualisiert – der häufigste Fall.
- **Läuft der Client**, fragt er den Benutzer: *jetzt aktualisieren* oder *um 1 / 4 / 16 Stunden verschieben*. **Beendet wird nur nach Zustimmung.** Schließt der Benutzer Ordivis Platform zwischenzeitlich von sich aus, nutzt der Dienst diesen Moment.
- **Keine UAC-Rückfrage:** Der Dienst läuft als `LocalSystem` und darf unter `C:\Programme` schreiben. Anwender **ohne lokale Administratorrechte** können ihren Client damit aktualisieren.

- **Nachprüfung:** Nach dem Dateitausch liest der Dienst die Version **an der EXE** erneut aus. Nur wenn sie der Zielversion entspricht, gilt der Lauf als erfolgreich; andernfalls wird der Snapshot zurückgespielt. Das Ergebnis meldet der Client beim nächsten Start.
- **Server-Adresse:** Der Dienst ermittelt sie aus denselben Quellen wie der Client, in dieser Reihenfolge: 1. `%ProgramData%\InfraDesk\connections.json` – das **Leitprofil** aus dem Verbindungsverzeichnis (siehe Abschnitt *Mehrere Installationen aus einem Client*). Diese Quelle ist maschinenweit und eindeutig; erreicht ein Arbeitsplatz mehrere getrennte Installationen, bestimmt sie, auf welchen Stand er gehoben wird. 2. `%LocalAppData%\InfraDesk\connection.json` – die im Client unter *Verbindungseinstellungen* zuletzt gewählte Adresse. Sie liegt im Benutzerprofil; der Dienst läuft als `LocalSystem` und durchsucht deshalb die Profile und nimmt das zuletzt geschriebene. 3. `<Installationsverzeichnis>\client\server-url.txt` – die Eingabe aus dem Setup. 4. `<Installationsverzeichnis>\client\debugConfig.json` – mitgelieferte Vorgabe (`http://localhost:5000`); auf einem eigenen Arbeitsplatz-PC ist sie fast immer falsch.

Welche Quelle gewonnen hat, steht im Protokoll. Fehlen die Punkte 1 bis 3, fragt der Dienst `localhost` und findet dort nichts – dann genügt es, `server-url.txt` mit der richtigen Adresse anzulegen und den Dienst neu zu starten. - **Protokoll:** `C:\ProgramData\InfraDesk\Logs\clientupdate-JJJJMMTT.log` sowie das Windows-Ereignisprotokoll (Quelle `InfraDesk.ClientUpdate`). Es nennt die verwendete Server-Adresse samt Herkunft, den Wartezustand vor dem Schließen des Clients und das Ergebnis der Nachprüfung.

> **Nach dem Update starten Sie Ordivis Platform wieder selbst** – der Dienst startet den Client bewusst nicht automatisch neu.

Rückfallweg ohne Dienst: Ist der Dienst nicht eingerichtet, nutzt der Client den mitgelieferten `InfraDesk.Updater.exe` (gleiche Prüfungen, gleicher Snapshot-Mechanismus). Liegt die Installation unter `C:\Programme`, verlangt dieser Weg eine **UAC-Rückfrage**; wird sie abgelehnt, bricht die Aktualisierung mit einer klaren Meldung ab.

> **Einmalig:** Clients, die **vor** diesem Release installiert wurden, bringen den Aktualisierungsdienst noch nicht mit. Sie müssen **einmal** über die Setup-Datei aktualisiert werden (Installationstyp „Client-Installation"); ab dann läuft die Aktualisierung von selbst.

Mehrere Installationen aus einem Client

Nicht jede Einrichtung führt alles auf einem Server. Wer mehrere getrennte Ordivis-Installationen betreibt – etwa Hauptverwaltung und Stadtwerke –, kann sie aus **einem** Client erreichen und auf der Anmeldemaske umschalten.

> ⚠️ **Nicht zu verwechseln mit Mandantenfähigkeit.** Dort teilen sich mehrere Mandanten *eine* Installation (ein Server, eine Registry, je Mandant eine Datenbank). Hier erreicht *ein Client mehrere getrennte Installationen* – jede mit eigenem Server, eigener Registry, eigener Version. Der Mandantenwechsler im Programm bleibt davon unberührt.

Fehlt die Datei, ändert sich nichts. Bestandsinstallationen laufen unverändert weiter, die Auswahl bleibt unsichtbar. Sie erscheint erst ab **zwei** Einträgen.

AUFBAU DER DATEI

Die Liste ist maschinenweit und liegt in `%ProgramData%\InfraDesk\connections.json`:

```
{
  "version": 1,
  "allowCustom": false,
  "connections": [
    { "id": "hv",  "name": "Hauptverwaltung", "apiBaseUrl": "https://ordivis.stadt.local:5001",
      "ignoreCertificateErrors": false, "useWindowsSso": true, "primary": true },
    { "id": "swk", "name": "Stadtwerke",      "apiBaseUrl": "https://sw.stadt.local:5001",
      "ignoreCertificateErrors": true,  "useWindowsSso": false }
  ]
}
```

Feld	Bedeutung
`id`	Kurze, gleichbleibende Kennung. Sie hält die Auswahl fest, auch wenn sich die Adresse ändert.
`name`	So erscheint die Installation auf der Anmeldemaske.
`apiBaseUrl`	Vollständige Adresse der API mit Schema und Port.
`ignoreCertificateErrors`	Zertifikatoption **dieser** Verbindung. Vorher galt sie für den ganzen Arbeitsplatz – bei einem Server mit gültigem und einem mit selbstsigniertem Zertifikat war das falsch.
`useWindowsSso`	Windows-Anmeldung **dieser** Verbindung. Produktivserver automatisch, Testserver nicht.
`primary`	Das **Leitprofil**. Genau ein Eintrag; fehlt die Marke, gilt der erste.
`allowCustom`	Ob daneben eine eigene Adresse zulässig ist. Vorgabe **aus** – die Liste gibt der Administrator vor.

DAS LEITPROFIL BESTIMMT DIE VERSION

Der Arbeitsplatz-Dienst richtet sich nach dem Leitprofil, nicht nach der zuletzt gewählten Verbindung. Ohne diese Festlegung zöge er den Arbeitsplatz auf die Version des **neuesten** Servers, mit dem er je verbunden war – und der Client spräche danach gegen eine ältere API, die er nicht kennt. Aus demselben Grund bietet der Veraltet-Hinweis im Client den Tausch **nur beim Leitprofil** an; bei den übrigen Verbindungen steht der Hinweis ohne Schaltfläche.

DATEIRECHTE

Ohne Schreibschutz gegen normale Benutzer wäre die Vorgabe eine Behauptung. Das Setup setzt die Rechte (Administratoren und `SYSTEM` Vollzugriff, Benutzer nur Lesen), schaltet die Vererbung ab und **liest sie zurück**. Auf einem Datenlaufwerk erbt der `ProgramData`-Zweig die Rechte nämlich nicht von selbst.

Prüfen:

```
Get-Acl "$env:ProgramData\InfraDesk\connections.json" | Format-List
```

WENN EINE FALSCHE VORGABE ALLE AUSSPERRT

Eine falsche Adresse in der Liste kann jeden aussperren, auch den Administrator. Es gibt zwei Wege zurück, beide vor Ort:

1. Die Datei auf dem Arbeitsplatz berichtigen (Administratorrechte nötig).
2. Das Setup erneut laufen lassen. ⚠️ Eine **bestehende** `connections.json` wird dabei **nicht** überschrieben – sie ist der gepflegte Bestand. Zum Zurücksetzen die Datei vorher löschen.

Wer die Tür grundsätzlich offen halten will, setzt `allowCustom` auf `true`; dann bleibt der Verbindungsdialog änderbar.

WAS DER WECHSEL VERWIRFT

Ein Wechsel ist kein Abmelden, räumt aber genauso auf: Sitzung, Live-Kanal, Rechte, Statusabfrage und die mandantenbezogenen Zwischenspeicher werden verworfen, und die Maske springt auf Schritt 1 zurück. Die abgelegte Windows-Anmeldung bleibt liegen – sie gehört seit diesem Release **je Verbindung** in eine eigene Datei, damit ein Erneuerungstoken niemals an den falschen Server geht.

Nicht angefasst werden gespeicherte Ticket-Ansichten und CMDB-Filter: Sie speichern Text, keine serverseitigen Kennungen, und sind serverübergreifend höchstens unpassend, nie falsch.

ANDROID-APP

Auf Android gibt es keinen `ProgramData`-Zweig. Die Liste kommt dort aus zwei Quellen, in dieser Reihenfolge:

1. **Android Managed Configuration** aus der MDM-Lösung – der richtige Weg für verwaltete Geräte, weil der Administrator sie ohne neuen Build pflegt. Die Felder erscheinen in der MDM-Konsole.
2. Die eingebettete `appsettings.json`, Abschnitt `Api:Connections` – sie kommt mit der verteilten Anwendung.

WEB-PORTAL

Das Portal bekommt bewusst **keine** Auswahl, sondern eine Linkliste. Es ist Blazor Server und legt seine API-Adresse beim Prozessstart einmalig fest; ein Portalprozess, der mit mehreren Schnittstellen spräche, wäre ein anderes Produkt. Die Liste steht in der `appsettings.json` unter `OtherInstallations` und führt in das jeweils eigene Portal der anderen Installation. Leere Liste = kein sichtbarer Unterschied.

5.1.2 BESTEHENDE ARBEITSPLÄTZE EINMALIG ANSTOSSEN (SEIT 2026.08.12.1127)

Bis zu dieser Fassung war der Aktualisierungsdienst die einzige ausgelieferte Komponente **ohne eigenen Update-Weg**: Das signierte Manifest kannte nur API, Portal, Client und Wallboard. Der Dienst hielt den ganzen Arbeitsplatz aktuell und blieb dabei selbst auf dem Stand seiner Installation stehen. Seit **2026.08.12.1127** ist `clientupdate` die **fünfte Komponente** im Manifest: Der Dienst prüft je Zyklus auch seine eigene Version am Artefakt und übergibt den Tausch dem Bootstrapper.

Diese Fähigkeit kann sich nicht selbst ausliefern. Ein Dienst ohne Selbstupdate-Code holt seinen Nachfolger prinzipbedingt nicht über das Delta – er kennt den Weg dorthin ja gerade nicht. Bereits eingerichtete Arbeitsplätze brauchen deshalb **genau einmal** einen Anstoß von außen. Ohne ihn bleibt der Aktualisierungsdienst auf altem Stand stehen, während er den Client weiter zuverlässig anhebt – ein Zustand, der von außen unauffällig aussieht. Danach hält der Dienst sich selbst aktuell, und der Handgriff wiederholt sich nie.

Dafür liegt `update-clientupdate.ps1` auf dem **Server** bereit, im Unterverzeichnis `scripts` Ihrer Installation – also neben `backup.ps1`. **Quelle** für die neuen Dateien ist `clientupdate` derselben Server-Installation: Eine Server-Installation trägt die Dateien des Aktualisierungsdienstes mit, registriert ihn dort aber bewusst nicht. Damit ist Ihr Server zugleich die Verteilstelle, und Sie müssen nichts zusätzlich bereitlegen.

Angenommen, die Installation liegt unter `D:\Program Files\Ordivis Platform`:

```
# Erst zeigen, was geschaehe – ohne eine Datei anzufassen
powershell -ExecutionPolicy Bypass -File "D:\Program Files\Ordivis Platform\scripts\update-clientup
```

```
# Dann ausfuehren, in einer Eingabeaufforderung MIT Administratorrechten
powershell -ExecutionPolicy Bypass -File "D:\Program Files\Ordivis Platform\scripts\update-clientup
```

Auf einem **entfernten** Arbeitsplatz geben Sie beides als Netzwerkpfad an – geben Sie `scripts` und `clientupdate` dazu auf dem Server frei, oder kopieren Sie beide Verzeichnisse einmal auf eine vorhandene Freigabe.

- **Bewusst nicht der Installer.** Das Skript fasst ausschließlich `<Installationsverzeichnis>\clientupdate` an: Dienst stoppen, Dateien tauschen, Dienst starten. **Keine Datenbank, kein PostgreSQL, kein** `%ProgramData%` – nichts von dem, was ein Installer-Lauf auf einer produktiven Anlage sonst berührt (Datenbank-Provisionierung, Bereinigung von Altbeständen). `appsettings*.json` bleibt liegen; eine angepasste Server-Adresse oder ein geändertes Prüfintervall überleben den Lauf.
- **Administratorrechte sind Pflicht.** Ein als `LocalSystem` laufender Dienst lässt sich nur erhöht stoppen, und unter `C:\Programme` darf ein gewöhnliches Konto nicht schreiben. Fehlen die Rechte, bricht das Skript **vor** der ersten Änderung ab, nicht auf halbem Weg.
- `-WhatIf` **zeigt den Lauf vorab:** installierte Version, Version der Quelle und Zielverzeichnis, ohne etwas zu ändern. Das Zielverzeichnis stammt aus der Dienst-Registrierung – geraten wird nichts. Mit `-InstallDir` lässt es sich vorgeben.
- **Der Nachweis steht am Artefakt, nicht am Exit-Code.** Nach dem Tausch liest das Skript die Version an der `InfraDesk.ClientUpdateService.exe` erneut aus und prüft, dass der Dienst wieder `Running` meldet. Stimmt eines von beidem nicht, endet der Lauf mit **Exit-Code 1** – wichtig, wenn Sie über mehrere Arbeitsplätze schleifen oder das Skript aus einer Verteilsoftware aufrufen.
- **Liegt bereits der neue Stand an**, meldet das Skript das und tut nichts. Ein versehentlicher zweiter Lauf schadet also nicht.

Welcher Handgriff gilt für Sie? Prüfen Sie auf dem Arbeitsplatz, was vorhanden ist – die erste Zeile nennt das Installationsverzeichnis, die zweite sagt, ob die Programmdateien liegen, die dritte, ob der Dienst eingerichtet ist:

```
$app = (Get-ItemProperty 'HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\*' | Where-Obje
```

Befund	Handgriff
Dienst vorhanden, aber älter als **2026.08.12.1127**	`update-clientupdate.ps1` – dieser Abschnitt
Dateien liegen, **Dienst fehlt**	`register-client-update-service.ps1`, ebenfalls in `scripts` – der Absatz unter dieser Tabelle
Dateien fehlen (Installation vor 2026.07.22.1615)	Setup-Datei, Installationstyp „Client-Installation"

Der mittlere Fall braucht nur die Registrierung, weil die Programmdateien bei jeder Client-Installation mitkommen – auch bei einer, die den Dienst damals noch nicht eingerichtet hat. Das Skript verlangt das Installationsverzeichnis und legt den Dienst genauso an wie das Setup (verzögerter Automatikstart, `LocalSystem`, Neustart nach Fehlern); ist er bereits vorhanden, wird er zuvor sauber entfernt:

```
# In einer Eingabeaufforderung MIT Administratorrechten
powershell -ExecutionPolicy Bypass -File "D:\Program Files\Ordivis Platform\scripts\register-client
```

`-InstallDir` ist das Verzeichnis auf dem **Arbeitsplatz** – auf einem Laptop üblicherweise unter `C:`, während der Server unter `D:` liegt. Das Skript bricht ab, statt einen Dienst anzulegen, der beim Start sofort scheitern würde, wenn dort keine `clientupdate\InfraDesk.ClientUpdateService.exe` liegt.

5.2 Update-Center & Auto-Update-Policy

- **3-Stufen-Policy** (persistiert in `SystemState`; Lesen `activation.manage`, Ändern `system.config.manage`): **Aus** / **Nur benachrichtigen** / **Vollautomatisch**. Bei Vollautomatik wendet der `ScheduledUpdateWorker` das signaturgeprüfte Delta selbsttätig im konfigurierten **Wartungsfenster** (Wochentage + Uhrzeit, 1-h-Fenster) an – mit Pre-Update-Backup, Health-Check und automatischem Rollback wie bei manueller Ausführung; der Lauf ist je Zielversion **dedupliziert** (`update.last_auto_applied_version`). Endpunkte `GET/PUT /api/v1/system/update/policy`.
- **Changelog-Quelle:** Die Website führt neben `version.json` eine fortgeschriebene `changelog.json` (je Release: `version`, `date`, `title`, `mandatory`, `components`, `changes` mit `category`/`text`). `GET /api/v1/system/changelog` liefert sie aufbereitet (10 min gecacht); der Publish-Dienst überträgt `changelog.json` wie `version.json` bei jedem Release.
- **Komponentenansicht:** je Komponente (API/Portal/Client/Wallboard) installierte Version, verfügbare Zielversion und „betroffen"-Kennzeichnung aus dem Delta-Plan – direkt im Client-Update-Center.

> **Zwei Rechner, eine Liste – auf die Beschriftung achten:** Alle Komponenten dieser Ansicht liegen auf dem **Server-Host**; die Zeile „Client (Kopie auf dem Server)" meint die Client-Installation *neben der API*, nicht Ihren Arbeitsplatz. Die eigene Installation steht getrennt darüber als **„Dieser Arbeitsplatz"**. Bei getrennten Installationen zeigen beide Zeilen zu Recht unterschiedliche Versionen.

6 Sicherung & Wiederherstellung

Die Datenbank wird server-seitig durch die geplante Windows-Aufgabe `InfraDesk-DailyBackup` gesichert (`pg_dump`). Ziel-Verzeichnis, Zeitplan und Aufbewahrung stellen Sie als **SuperAdmin** unter *Einstellungen → Backup* ein; von dort lässt sich eine Sicherung auch sofort anstoßen und eine vorhandene Sicherung wiederherstellen.

6.1 Verschlüsselte Sicherungen (7z, AES-256)

> 💻 **Nur im Client:** *Fußbereich ▸ Einstellungen ▸ Backup* · 🔑 `system.backup.manage`

Ist ein **Sicherungs-Kennwort** hinterlegt, werden die Sicherungen als **AES-256-verschlüsselte 7z-Archive** (`ordivis_<Zeitstempel>.7z`) abgelegt; der unverschlüsselte Zwischenstand wird nach dem Verschlüsseln gelöscht. Ohne Kennwort bleibt es bei einer unverschlüsselten `.dump`-Datei (die Oberfläche weist darauf hin).

🔒 **Pflichtreihenfolge** — hinterlegen Sie das Kennwort **an einem zweiten, sicheren Ort**, bevor Sie es hier speichern. Ohne das Kennwort ist eine verschlüsselte Sicherung wertlos, und es gibt keinen Weg daran vorbei.

1. Legen Sie das gewählte Kennwort im Kennwortspeicher Ihres Hauses ab.
2. Öffnen Sie *Einstellungen ▸ Backup*.
3. Tragen Sie das **Sicherungs-Kennwort** ein.

4. Speichern Sie.
5. Warten Sie den nächsten Sicherungslauf ab und prüfen Sie, dass eine `.7z`-Datei entsteht.

✅ **Ergebnis:** Ab der nächsten Sicherung entstehen verschlüsselte Archive.

- Das Kennwort gilt **zentral** für die Installation. Ein **leeres** Feld beim Speichern lässt ein bereits hinterlegtes Kennwort unverändert; ein neuer Wert ersetzt es und wirkt ab der nächsten Sicherung.
- **Wiederherstellung:** Beim Zurückspielen eines verschlüsselten Archivs wird das Kennwort **abgefragt** (nicht aus der Konfiguration übernommen). So lässt sich auch eine ältere Sicherung mit abweichendem Kennwort zurückspielen. Unverschlüsselte `.dump`-Sicherungen werden weiterhin ohne Abfrage wiederhergestellt.

Achtung: Ohne das Kennwort lässt sich eine verschlüsselte Sicherung **nicht** wiederherstellen – es gibt keine Hintertür. Bewahren Sie das Kennwort sicher und **getrennt von den Sicherungen** auf.

Wichtig: Das Sicherungs-Kennwort wird **niemals im Klartext** abgelegt. Es wird mit **Windows-DPAPI** verschlüsselt in `operations.json` gespeichert und lässt sich nur im Dienst-/Aufgabenkontext (SYSTEM) auf **diesem** Server entschlüsseln. Wird die Datei kopiert oder von einem anderen Konto gelesen, bleibt das Kennwort unbrauchbar.

6.2 Weitere Sicherung & Wiederherstellung

- **Konfiguration & Secrets:** `appsettings` (ohne Geheimnisse) und den **Vault-Schlüssel** sicher und getrennt aufbewahren.
- **Wiederherstellung:** Datenbank zurücksichern, Dienste konfigurieren, Vault-Schlüssel bereitstellen.

Achtung: Ohne den originalen **Vault-Schlüssel** lassen sich verschlüsselte Zugangsdaten (Discovery, E-Mail) nach einer Wiederherstellung nicht entschlüsseln und müssen neu erfasst werden.

6.3 Anhänge liegen in der Datenbank

Anhänge – das Bildschirmfoto an einem Vorgang, die archivierte Roh-E-Mail, ein Dokument im BCM – werden **in der Datenbank** gespeichert. Sie sind damit von der Sicherung aus 6.1 **mit erfasst**; ein zweiter Sicherungsweg für ein Dateiverzeichnis entfällt, und eine zurückgespielte Datenbank bringt ihre Anhänge mit.

Achtung: Bis einschließlich **2026.08.12** war das anders. Die Anhänge lagen als Dateien im **Temp-Verzeichnis des Dienstkontos** – ein Ablageort, den die Datenträgerbereinigung leert, den `pg_dump` nicht erfasst und der sich mit dem Dienstkonto verschiebt. Die Zeile in der Datenbank überlebte das jedes Mal, der Inhalt nicht. Prüfen Sie nach dem Update das Protokoll der Übernahme, **bevor** Sie sich auf eine ältere Sicherung verlassen: Was dort schon fehlte, kann keine Übernahme zurückholen.

So prüfen Sie die Übernahme nach dem Update:

🔑 Lesezugriff auf das Serverprotokoll — nicht in Client oder Portal sichtbar.

🔒 **Pflichtreihenfolge** — prüfen Sie das Protokoll, **bevor** Sie sich auf eine ältere Sicherung verlassen. Was dort schon fehlte, kann keine Übernahme zurückholen.

1. Starten Sie den Dienst nach dem Update und warten Sie den ersten Lauf ab.
2. Öffnen Sie das Protokoll (Kapitel 10).
3. Lesen Sie den Eintrag zur Anhang-Übernahme.

✅ **Ergebnis:** Sie wissen, wie viele Anhänge übernommen wurden und welche gefehlt haben.

Die Übernahme läuft **je Datenbank einmal**, danach ist nichts mehr zu tun. Das Protokoll sagt, was passiert ist:

```
Anhänge: 30 Altbestand-Dateien werden in die Datenbank übernommen. Fundorte: …
Anhänge: 30 Altbestand-Dateien vollständig übernommen.
```

Gesucht wird der Reihe nach in `Attachments:RootPath` (falls gesetzt), in `Attachments:LegacyRootPaths`, im Temp-Verzeichnis des laufenden Dienstkontos sowie in `%SystemRoot%\SystemTemp` und `%SystemRoot%\Temp` – jeweils im Unterverzeichnis `infradesk-attachments`. Damit wird auch der Fall gefunden, dass der Dienst inzwischen unter einem anderen Konto läuft als beim Hochladen.

Diese Zusagen gelten für den Lauf:

- **Nichts wird gelöscht.** Die Altdateien bleiben liegen. Sie sind das Sicherheitsnetz, solange niemand die Übernahme geprüft hat; aufräumen können Sie danach von Hand.
- **Eine nicht gefundene Datei hält nichts auf.** Sie wird als Warnung gemeldet, die übrigen Anhänge werden trotzdem übernommen, und der Start bricht nicht ab.
- **Ein Fehlschlag ist wiederholbar.** Wo die Altdatei fehlte, behält die Zeile ihren Pfad. Tragen Sie den tatsächlichen Ablageort unter `Attachments:LegacyRootPaths` nach (mehrere durch **Semikolon** getrennt) und starten Sie den Dienst neu – der nächste Lauf holt sie nach.

```
{ "Attachments": { "LegacyRootPaths": "E:\\Gesichert\\infradesk-attachments;F:\\Alt\\infradesk-atta
```

So zählen Sie jederzeit nach, was noch aussteht (0 = vollständig übernommen):

```
SELECT count(*) FROM discovery.attachments WHERE storage_path IS NOT NULL;
```

7 Sicherheit & Härtung

- **HTTPS** für API und Portal einrichten (siehe 7.1) – für den Produktivbetrieb **dringend empfohlen**.
- **PostgreSQL** härten: nur benötigte Netze, starke Passwörter, App-Rolle mit minimalen Rechten.
- **Dienstkonten** mit minimalen Rechten betreiben.
- **AD/LDAP-Anbindung** ist read-only.
- Details und Kontrollrahmen: ISMS-Modul (ID-28).

7.1 HTTPS einrichten

> **Sicherheitshinweis:** Ohne HTTPS laufen Anmeldung und sämtliche Daten **unverschlüsselt** über das Netz. Für den Produktivbetrieb ist HTTPS für API **und** Portal dringend zu verwenden.

Ordivis Platform kennt **drei Betriebsarten** der Transportverschlüsselung (Abschnitt `Hosting`, gleich strukturiert für API und Portal). Sie sind bei der Installation wählbar und lassen sich jederzeit als **SuperAdmin** unter *Einstellungen*

→ *Plattform* → *HTTPS / Transportsicherheit* ändern. Weil Kestrel die Endpunkte nur beim Start liest, wird eine Änderung **erst nach einem Neustart** des Dienstes `InfraDesk.API` wirksam (der SuperAdmin-Dialog weist darauf hin).

Der SuperAdmin-Wunschzustand wird in `C:\ProgramData\InfraDesk\config\hosting.json` abgelegt. Diese Datei liegt **außerhalb** des Installationsverzeichnisses und **überlebt Produkt-Updates** (im Gegensatz zur mitgelieferten `appsettings.json`).

Betriebsart	Wann?	Was passiert
Reverse-Proxy (Standard)	TLS terminiert ein vorgelagerter Proxy (IIS/nginx/Traefik)	Der Dienst bindet nur HTTP; ein direkter, unverschlüsselter Zugriff auf den Port ist ein Risiko.
Eigenes Zertifikat	PEM-Text einfügen, vorhandene PFX-Datei oder Windows-Zertifikat	Kestrel bindet HTTP **und** HTTPS mit diesem Zertifikat.
Let's Encrypt (ACME)	öffentlich erreichbarer Server mit Domäne	Zertifikat wird automatisch bezogen und erneuert (HTTP-01-Challenge über Port 80).

a) Eigenes Zertifikat – als Text einfügen (empfohlen):

Zertifikate aus einer CA-Oberfläche (OPNsense, XCA, interne bzw. Behörden-PKI) liegen fast immer als **PEM** vor: eine Zertifikatsdatei (`*.crt`), eine Schlüsseldatei (`*.key`) und meist eine Kettendatei. Diese drei Inhalte lassen sich direkt einfügen; die Umwandlung übernimmt der Server.

1. SuperAdmin-Dialog → *HTTPS / Transportsicherheit* → Betriebsart *Eigenes Zertifikat* → Abschnitt **„Zertifikat als Text einfügen (PEM)"**.
2. Inhalt der Dateien in die drei Felder kopieren. Ist der private Schlüssel verschlüsselt (`-----BEGIN ENCRYPTED PRIVATE KEY-----`), zusätzlich sein Kennwort angeben.
3. **„Prüfen und übernehmen"**. Es wird sofort gemeldet, ob Zertifikat und Schlüssel zusammenpassen, bis wann es gilt und **für welche Namen** – bei einem Fehler steht der Grund direkt dort.
4. Domäne und HTTPS-Port kontrollieren, speichern → Dienst `InfraDesk.API` neu starten.

Die Ausstellerkette ist optional, aber **empfohlen**: Der Server schickt sie beim Verbindungsaufbau mit. Ohne sie funktioniert es häufig, doch manche Klienten – und nahezu jedes Mobilgerät – lehnen die Verbindung ab.

Was mit dem Schlüssel geschieht: Aus den eingefügten Inhalten erzeugt der Server eine PFX-Datei unter `C:\ProgramData\InfraDesk\config\certs\`, die **nur** SYSTEM und Administratoren lesen dürfen. Ihr Kennwort wird zufällig vergeben und verschlüsselt hinterlegt – Sie müssen es weder kennen noch eingeben. Der private Schlüssel liegt damit an keiner Stelle im Klartext.

Der Name im Zertifikat entscheidet: Maßgeblich ist der *alternative Name* (SAN), **nicht** der Common Name – kein heutiger Browser wertet den CN noch aus. Ein brauchbares Zertifikat trägt den Hostnamen, unter dem der Server aufgerufen wird; sinnvollerweise zusätzlich seine IP-Adresse, damit der Zugriff auch bei DNS-Störungen ohne Warnung funktioniert. Fehlt der SAN, sagt die Übernahme das ausdrücklich.

a2) Eigenes Zertifikat – vorhandene PFX-Datei:

1. PFX (mit privatem Schlüssel) auf den Server legen, z. B. `C:\ProgramData\InfraDesk\certs\server.pfx`.
2. SuperAdmin-Dialog: Betriebsart *Eigenes Zertifikat*, Pfad + Kennwort eintragen (alternativ ein Thumbprint aus `LocalMachine\My`), Domäne und HTTPS-Port (Standard 5001) setzen.

3. Speichern → Dienst `InfraDesk.API` neu starten.

b) Let's Encrypt / ACME:

1. Voraussetzung: Der Server ist unter der Domäne **aus dem Internet** erreichbar und **Port 80** ist offen (Let's Encrypt prüft die Domäne über `http://{Domäne}/.well-known/acme-challenge/…`).
2. SuperAdmin-Dialog: Betriebsart *Let's Encrypt*, Domäne + Kontakt-E-Mail eintragen, Nutzungsbedingungen akzeptieren. **Test-Umgebung (Staging)** zunächst eingeschaltet lassen – sie umgeht die strengen Mengen-Grenzen; das Zertifikat ist dann aber **nicht** öffentlich vertrauenswürdig.
3. Speichern → Dienst neu starten. Das Zertifikat wird beim Start im Hintergrund bezogen; die Erneuerung läuft automatisch (Prüfung alle 12 h, Erneuerung ab 30 Tagen vor Ablauf) **ohne** weiteren Neustart.
4. Nach erfolgreichem Test **Staging ausschalten** und erneut starten, um ein produktives Zertifikat zu beziehen.

c) Reverse-Proxy (Standard): Der Dienst bleibt auf HTTP; die Verschlüsselung übernimmt der vorgelagerte Proxy. Den HTTP-Port dann **nicht** direkt aus dem Netz erreichbar machen (nur der Proxy spricht ihn an).

d) Self-Service-Portal – dasselbe Zertifikat, eigene Ports:

Das Portal ist ein eigener Windows-Dienst und braucht deshalb **eigene Ports** – zwei Dienste können denselben Port nicht binden. Betriebsart, Domäne und **Zertifikat teilt es sich mit der API**; ein zweites Zertifikat ist nicht nötig.

1. SuperAdmin-Dialog → Abschnitt „**Ports des Self-Service-Portals**": HTTPS-Port auf **443** setzen (Vorgabe ist `0` = kein eigenes HTTPS fürs Portal).
2. Speichern → Dienst `InfraDesk.Portal` neu starten.
3. **Firewall-Freigabe für Port 443 anlegen**, falls das Portal-HTTPS erst nachträglich eingeschaltet wurde – der Installer legt sie nur an, wenn beim Installieren schon ein HTTPS-Modus gewählt war:

```
powershell New-NetFirewallRule -DisplayName "Ordivis Portal (HTTPS)" -Direction Inbound ` -Protocol TCP -LocalPort 443 -Action Allow -Profile Any
```

> **Warum 443 empfohlen ist:** Das Portal ist die Adresse, die Sie an die Belegschaft weitergeben. Auf dem Standard-Port ist es ohne Portangabe erreichbar (`https://helpdesk.ihre-domäne.de`). Die API bleibt auf 5001 – sie wird nicht von Hand aufgerufen.
>
> **Links in E-Mails** (Kennwort zurücksetzen, Termin zusagen, Bewertung) werden aus Domäne und Portal-Port **abgeleitet**, sobald HTTPS mit einer Domäne eingerichtet ist. Steht das Portal auf 443, enthalten die Mails saubere Adressen ohne Portangabe.
>
> **Let's Encrypt und das Portal:** Das Portal bezieht **kein eigenes** Zertifikat – die Domänenprüfung verlangt Port 80, und den kann nur einer der beiden Dienste halten. Es übernimmt das von der API bezogene Zertifikat automatisch, auch nach jeder Erneuerung.

7.2 Client-Verbindung (API-Adresse)

Der Windows-Client muss wissen, unter welcher Adresse die API erreichbar ist – in der Regel laufen API und Datenbank auf **einem** Server. Diese Adresse wird bei der **Client-Installation** abgefragt und ist danach jederzeit änderbar:

- **Auf der Anmeldeseite** über *Verbindungseinstellungen* (auch **vor** der Anmeldung – eine falsche Adresse ließe sich sonst nicht korrigieren).
- **In der Anwendung** als SuperAdmin unter *Einstellungen → Plattform → API-Verbindung*.

Die Adresse (z. B. `https://server.firma.local:5001`) wird **lokal je Arbeitsplatz** gespeichert (nicht am Server, da sie schon vor der Anmeldung gebraucht wird) und beim nächsten Anmelden wirksam. Mit `https://` verbindet sich der Client verschlüsselt; die Schaltfläche *Verbindung testen* prüft die Erreichbarkeit.

Zertifikatfehler ignorieren (selbstsigniertes / internes Zertifikat): Nutzt der Server ein selbstsigniertes oder ein von einer internen CA ausgestelltes Zertifikat, das der Arbeitsplatz **nicht als vertrauenswürdig** kennt, würde der Client die HTTPS-Verbindung normalerweise ablehnen. Für kleine Installationen ohne eigene PKI gibt es im Verbindungsdialog die Option **„Zertifikatfehler bei HTTPS ignorieren"**. Damit ist die Verbindung **verschlüsselt, aber nicht auf Echtheit geprüft** – besser als gar kein HTTPS, aber ein bewusster Kompromiss (theoretisch Man-in-the-Middle möglich). Der Client zeigt in diesem Fall dauerhaft den Hinweis *„Zertifikatprüfung deaktiviert"* an.

> **Richtig gemacht** (empfohlen, statt Fehler dauerhaft zu ignorieren): Ein **vertrauenswürdiges** Zertifikat verwenden – entweder von einer öffentlichen CA / Let's Encrypt (siehe 7.1) **oder** das Zertifikat der **internen CA** einmalig auf den Arbeitsplätzen als vertrauenswürdig hinterlegen (Zertifikat der Stammzertifizierungsstelle in den Windows-Speicher *Vertrauenswürdige Stammzertifizierungsstellen* importieren, z. B. per Gruppenrichtlinie). Danach die Option wieder abschalten. Für das **Portal→API** gibt es dieselbe Möglichkeit über die Portal-Einstellung `ApiIgnoreCertificateErrors: true`.

7.3 TLS-Version

Ordivis Platform gibt die TLS-Version **nicht fest vor**, sondern überlässt sie dem Betriebssystem (`SslProtocols.None` – von Microsoft empfohlen, damit das OS die beste verfügbare Version wählt und über Windows-Updates gepflegt wird). In der Praxis heißt das: **TLS 1.3 wird automatisch genutzt, sobald das Server-Betriebssystem es unterstützt** (Windows 11 / Windows Server 2022 und neuer), sonst **TLS 1.2** als Rückfall. Ältere Versionen (TLS 1.0/1.1) sind auf aktuellen Windows-Systemen bereits per Schannel-Standard deaktiviert. Eine strengere Mindestversion lässt sich bei Bedarf über die **Schannel-Härtung des Servers** (Registry/Gruppenrichtlinie) erzwingen.

7.4 Zugänge für Geräte und Programme (Token)

Nicht jeder Zugriff auf Ordivis Platform kommt von einem Menschen an einer Tastatur. Ein Scan-Werkzeug im Nebenstandort und ein Infobildschirm im Empfangsbereich brauchen beide einen dauerhaften Zugang – aber für keinen von beiden darf ein Kennwort auf einem fremden Rechner liegen. Dafür gibt es **Token**: langlebige Anmelde-Merkmale, die einzeln benannt und einzeln widerrufen werden können.

Alle Token verwalten Sie an einer Stelle: **Einstellungen → Plattform → Integrations-Token**. Die Seite trennt sie in zwei Abschnitte, weil sie unterschiedlich funktionieren:

	Externe Programme	Anzeigegeräte (Wallboard)
Wofür	Werkzeuge, die Daten per Schnittstelle einspeisen	Bildschirme, die unbeaufsichtigt Kennzahlen zeigen
Wer ist der Zugriff	der Token selbst – kein Benutzerkonto	ein **Benutzerkonto**, an das der Token gebunden ist
Rechte	fest auf das Einspeisen von Scan-Ergebnissen begrenzt	die Rechte des gebundenen Kontos
Laufzeit	freiwillig	**Pflicht** (1 bis 730 Tage)

Gemeinsam ist beiden: Der **rohe Token wird nur einmal bei der Ausstellung angezeigt**; gespeichert wird nur sein Hash. Geht er verloren, stellen Sie einen neuen aus und widerrufen den alten.

> **Berechtigung:** Der Abschnitt „Externe Programme" verlangt `platform.integrationtoken.manage`, der Abschnitt „Anzeigegeräte" `system.config.manage`. Wer nur eines der beiden Rechte hält, sieht auch nur den zugehörigen Abschnitt.

7.4.1 EXTERNE PROGRAMME (SCAN-INGEST)

Externe Administrations-Werkzeuge – etwa das kostenlose **Ordivis Toolbox** – können Netzwerkscan-Ergebnisse über eine Schnittstelle in die **IPAM** einspeisen. Das ist besonders für **entfernte Subnetze** nützlich, die der Ordivis-Server selbst nicht erreicht (andere Standorte, isolierte Segmente): Das Werkzeug läuft auf einem Admin-Arbeitsplatz mit Sicht auf das Netz und übermittelt die Ergebnisse an Ordivis Platform. Für diese Anbindung wird **kein Benutzer-Login** verwendet, sondern ein **Integrations-Token** (Maschinen-Zugang).

Token ausstellen (nur SuperAdmin):

> 💻 **Nur im Client:** *Fußbereich* ▸ *Einstellungen* ▸ *Integrations-Token* · 🔑 `tenant.manage`

🔒 **Pflichtreihenfolge** — der rohe Token wird **einmal** angezeigt. Halten Sie das Zielsystem bereit, bevor Sie ihn ausstellen.

1. Im Client *Einstellungen* ▸ *Integrations-Token* öffnen, Abschnitt **Externe Programme**.
2. **Token für Programm** wählen, einen sprechenden **Namen** vergeben (z. B. „Ordivis Toolbox – Standort Nord") und optional ein **Ablaufdatum** setzen.
3. Der **rohe Token wird nur einmalig angezeigt** – kopieren und im externen Werkzeug hinterlegen. Danach ist er nicht mehr abrufbar (gespeichert wird nur ein Hash). Bei Verlust einen neuen ausstellen.

Eigenschaften und Grenzen:

- Ein Token gilt für **genau einen Mandanten** und trägt nur die **Einspeise-Rechte** – keinen allgemeinen Lese- oder Schreibzugriff auf andere Daten und keinen Zugang zur Oberfläche.
- Token lassen sich jederzeit **widerrufen**; ein Agent, der einen widerrufenen Token nutzt, wird sofort mit **401** abgewiesen.
- Eingespeiste Daten werden **nicht blind übernommen**: Neue Geräte erscheinen als **unbestätigte** IP-Adressen und neu angelegte Subnetze im Status **„Reserviert"**. Eine Administratorin prüft und bestätigt sie anschließend in der IPAM.
- **Rechte wachsen einem Token nicht nachträglich zu.** Ein vor einer Erweiterung ausgestellter Token erreicht die neuen Endpunkte nicht (Antwort **403**) – dafür muss ein neuer Token ausgestellt werden. Das ist beabsichtigt: Was ein Maschinen-Zugang darf, wird bei seiner Ausstellung entschieden.

Was ein Werkzeug einspeisen kann:

Datenquelle	Endpunkt	Landet als
Netzwerkscan	`POST /api/v1/ipam/ingest/scan`	unbestätigte IP-Adressen, Subnetz „Reserviert"
DHCP-Bereiche	`POST /api/v1/ipam/ingest/dhcp`	Subnetze + Reservierungen (Vorschau möglich)

Datenquelle	Endpunkt	Landet als
DNS-/PTR-Abgleich	`POST /api/v1/ipam/ingest/dns`	Namen an bekannten Adressen; Widersprüche werden **gemeldet**, nicht bereinigt
Geräte, AD-Computer, Software	`POST /api/v1/discovery/ingest/devices`	Funde im **Client-Inventar** (Prüfstand)
Zertifikate	`POST /api/v1/discovery/ingest/certificates`	CI der Klasse „Zertifikat" im Zustand **Staging**, nicht verifiziert
Schwachstellen (CVE/WID)	`POST /api/v1/security/ingest/findings`	Risiken in der Analyse „Schwachstellen (Scan)" des Informationsverbunds

Zwei Dinge kommen dabei **nie** an: **LAPS-Kennwörter** und **BitLocker-Wiederherstellungsschlüssel**. Die Werkzeuge übertragen sie nicht, auch wenn sie sie im Verzeichnis sehen könnten.

Erreichbarkeit: Alle Ingest-Endpunkte laufen über dasselbe HTTPS wie die übrige API (siehe **7.1 HTTPS einrichten**). Verbindungstests: `GET /api/v1/ipam/ingest/health` und `GET /api/v1/discovery/ingest/health`; die Werkzeuge bieten dafür üblicherweise eine Schaltfläche „Verbindung testen".

> **Sicherheitshinweis:** Behandeln Sie Integrations-Token wie Passwörter. Stellen Sie pro Werkzeug/Standort einen eigenen Token aus (bessere Nachvollziehbarkeit und gezielter Widerruf) und vergeben Sie bei Bedarf ein Ablaufdatum.

7.4.2 ANZEIGEGERÄTE (WALLBOARD)

Ein **Wallboard** ist ein Bildschirm, der unbeaufsichtigt läuft und im Wechsel Kennzahlen zeigt – im Service-Desk-Büro, am Empfang, im Leitstand. Das Gerät soll nach einem Stromausfall von allein wieder hochkommen, ohne dass jemand ein Kennwort eintippt. Genau dafür ist dieser Token da; die Bedienung des Bildschirms selbst beschreibt das Kapitel **Reporting & Dashboards**.

Token ausstellen:

> 💻 **Nur im Client:** *Fußbereich* ▸ *Einstellungen* ▸ *Integrations-Token* · 🔑 `tenant.manage`

🔒 **Pflichtreihenfolge** — auch hier wird der rohe Token nur **einmal** angezeigt.

1. *Einstellungen* ▸ *Integrations-Token* öffnen, Abschnitt **Anzeigegeräte (Wallboard)**.
2. **Token für Anzeigegerät** wählen und ausfüllen: - **Name des Anzeigegeräts** – wo der Schirm hängt (z. B. „Empfang EG"). Danach wird er einmal gezielt widerrufen, wenn das Gerät ausgetauscht wird. - **Konto, unter dessen Rechten der Schirm liest** – nur aktive Konten stehen zur Auswahl. - **Laufzeit in Tagen** – Vorgabe 365, höchstens 730.
3. Der **rohe Token wird nur einmalig angezeigt** – kopieren und auf dem Anzeigegerät in der Wallboard-App unter **Einrichtung → Wallboard-Token** eintragen.

Was der Schirm sehen darf, entscheidet das Konto. Der Token trägt keine eigenen Rechte; jede Anfrage wird mit den Rechten des gebundenen Kontos beantwortet.

Sie können einen Token nur für ein Konto ausstellen, dessen Rechte Sie selbst vollständig haben. Seit Version 2026.07.28.1912 gilt hier dieselbe Schranke wie bei der Rollenvergabe. Der Grund: Ein Wallboard-Token wird **anonym und ohne zweiten Faktor** gegen ein Zugriffstoken mit den Rechten des gebundenen Kontos eingetauscht. Ohne diese Schranke hätte, wer Anzeigegeräte verwalten darf, sich über ein vorgebliches „Wallboard" die Rechte eines höher privilegierten Kontos verschaffen können – ohne Kennwort und ohne dass eine Rollenzuweisung protokolliert worden wäre. Steht ein Konto nicht zur Auswahl, fehlen Ihnen selbst Rechte, die dieses Konto hat.

Empfehlung: Legen Sie für Wallboards ein **eigenes Konto mit ausschließlich lesenden Rechten** an und verwenden Sie **nicht** ein persönliches Konto. Ein Bildschirm im Flur zeigt allen Vorbeigehenden, was dieses Konto sehen darf – und ein gestohlener Token gäbe genau diese Sicht in fremde Hände.

Was beim Widerruf passiert: Der Schirm zeigt beim nächsten Erneuern der Sitzung einen Hinweis statt Zahlen. Ein laufendes Wallboard fällt also nicht sofort aus, aber innerhalb kurzer Zeit.

Ablage auf dem Gerät: Die Wallboard-App speichert den Token mit **Windows-DPAPI**, gebunden an das Windows-Konto, unter dem sie eingerichtet wurde. Läuft das Wallboard später unter einem anderen Windows-Konto, muss der Token dort neu eingetragen werden – die alte Datei lässt sich nicht entschlüsseln.

8 Skripte & Encoding

Achtung: PowerShell-Skripte (`.ps1`) mit Sonderzeichen **müssen UTF-8 mit BOM** sein, sonst Parse-Fehler unter PowerShell 5.1 auf dem Server.

9 Fehlerbehebung (Auszug)

Symptom	Ursache / Abhilfe
Dienst startet nach Update nicht	SCM-Timeout → erneut `Start-Service`; ggf. Prozess elevated killen
404 statt 401 nach Update	DLLs gesperrt (Dienst nicht rechtzeitig gestoppt) → `taskkill`, neu deployen
API-Fehler 42501 / DbUpdate	Tabellen gehören `postgres` → `ALTER TABLE ... OWNER TO infradesk_app`
Client-Absturz nach Splash	fehlende PRI (Publish statt Build) oder fehlende StaticResource → `%LOCALAPPDATA%\InfraDesk\Logs\client-error.log` (das Startprotokoll aus der Zeit, bevor das reguläre Protokoll läuft)
E-Mail-Versand scheitert	Root-CA fehlt / VPN nötig / `Notifications:Smtp` prüfen
Discovery entschlüsselt nicht	Vault-Schlüssel passt nicht zu Geheimnissen → Zugangsdaten neu hinterlegen
500 bei Zeitwerten	DateTimeOffset muss vor dem Speichern `.ToUniversalTime()` sein (timestamptz)

Symptom	Ursache / Abhilfe
Update meldet „kein gültig signiertes Manifest"	**Nicht der Meldung glauben, sondern das Update-Protokoll lesen** (`update-logs\` , siehe 10.4). Steht dort „Staging": Das Manifest ist in Ordnung – es fehlen Dateien auf dem Update-Server. Welche, sagt `api-JJJJMMTT.log` .
Update meldet „Es läuft bereits ein Update-Vorgang"	Es läuft immer nur **ein** Update gleichzeitig. Die Meldung nennt seit wann – und, falls der Vorgang abgebrochen ist, wann die Sperre von selbst freigibt (30 Minuten nach dem Start) sowie den Pfad der Statusdatei. Ein zweiter Anlauf davor wird abgewiesen; das ist kein Fehler.
LDAP-Sync meldet nur „fehlgeschlagen"	Ursache steht als innere Ausnahme im `api-JJJJMMTT.log` . Binäre Verzeichnisattribute werden beim Abgleich übersprungen (siehe Kapitel 41, Abschnitt 8.1) – schlägt der Sync **jedes Mal** fehl, ist die Ursache in aller Regel die Verbindung oder das Konto, nicht ein einzelner Datensatz.

10 Logs

Jede Komponente schreibt ihre eigene Logdatei – und zwar lokal auf dem Rechner, auf dem sie läuft. Der Dateiname nennt die Quelle, man sieht also auf einen Blick, wer geschrieben hat:

Datei	Anwendung	Wo	Dienst
`api-JJJJMMTT.log`	API / Anwendungsserver	**Server**, `C:\ProgramData\InfraDesk\Logs`	`InfraDesk.API`
`portal-JJJJMMTT.log`	Self-Service-Portal	**Server**, ebd.	`InfraDesk.Portal`
`worker-JJJJMMTT.log`	Discovery-Collector	**Rechner im Zielnetz**, ebd.	`InfraDesk.Discovery.Collector`
`updateagent-JJJJMMTT.log`	Update-Agent (Mehr-Server-Betrieb)	**Server**, ebd.	`InfraDesk.UpdateAgent`
`updater-JJJJMMTT.log`	Aktualisierung (Arbeiter + Fortschrittsfenster)	**Server / Arbeitsplatz**, ebd.	–
`clientupdate-JJJJMMTT.log`	Client-Update-Dienst	**Arbeitsplatz**, ebd.	`InfraDesk.ClientUpdate`
`client-JJJJMMTT.log`	Windows-Client	**Arbeitsplatz**, `%LOCALAPPDATA%\InfraDesk\Logs`	–
`wallboard-JJJJMMTT.log`	Wallboard	**Anzeigegerät**, ebd.	–
`mobile-JJJJMMTT.log`	Mobile App (Android)	**Mobilgerät**, im privaten Datenbereich der App	–

Die Dateien **rollieren automatisch**: im eingestellten Takt ein neuer Stand, zusätzlich bei Überschreiten der Größe. Alte Stände über die eingestellte Anzahl hinaus werden selbsttätig gelöscht – die Logs können also nicht unbemerkt die Platte füllen.

Warum der Client ins Benutzerprofil schreibt: Er läuft im Kontext des angemeldeten Benutzers und hat unter `ProgramData` je nach Installation kein Schreibrecht. Außerdem würden sich auf einem Terminalserver sonst alle Benutzer in dieselbe Datei schreiben. Dasselbe gilt für Wallboard und App.

10.1 Einstellen im Client (empfohlener Weg)

Der Weg über die Oberfläche ist der **maßgebliche**: **Einstellungen → Protokolle (Logs)**. Was dort gespeichert wird, liegt in `%ProgramData%\InfraDesk\config\operations.json`, überlebt ein Produkt-Update und gilt für API, Portal und Collector gemeinsam. Der Windows-Client holt sich seine Stufe beim Anmelden von dort ab. Die Seite verlangt die Plattform-Berechtigung `platform.hosting.manage` (SuperAdmin).

Einstellbar sind:

Bereich der Seite	Was sich einstellen lässt
Protokollierung	Hauptschalter über alle Anwendungen.
Speicherort	Zielordner der Server-Dienste. Empfehlung: nicht auf `C:` , sondern auf ein Datenlaufwerk.
Ausführlichkeit	Mindeststufe: `Debug` · `Information` · `Warnung` · `Fehler` .
Rotation & Aufbewahrung	Takt (stündlich / täglich / monatlich / jährlich / nie), Anzahl aufbewahrter Dateistände, Größe für einen zusätzlichen Umbruch, Format (Text oder JSON/CLEF).
Anwendungen	Ein/Aus je Anwendung – und optional eine eigene Stufe für genau eine davon.
Was genau protokolliert wird	Mindeststufe je Bereich (Technik, Anwendungen, Fachmodule) – etwa „Datenbank (SQL)" auf `Debug` , während alles andere ruhig bleibt.

Was sofort wirkt und was nicht. Stufen und Schalter greifen **ohne Dienst-Neustart** – genau darauf kommt es an, wenn gerade jemand am Telefon ist: umstellen, den Vorgang wiederholen, zurückstellen. Speicherort, Rotationstakt, Aufbewahrung und Format legt die Anwendung beim Start fest; sie wirken erst beim nächsten Start. Die Seite sagt nach dem Speichern, welcher der beiden Fälle eingetreten ist.

Fehler kommen immer durch. Auch bei Stufe „Fehler" wird jeder Fehler geschrieben – die Stufe legt nur fest, wie viel *zusätzlich* protokolliert wird. Für eine Fehlersuche vorübergehend auf `Debug` stellen, danach zurückstellen: Auf `Debug` wachsen die Dateien schnell.

„Debug" macht die Laufzeitumgebung absichtlich nicht gesprächig. Die technischen Bereiche (`.NET` , Datenbank, HTTP) bleiben auch dann auf „Warnung" – sonst stünde in jeder zweiten Zeile eine Datenbankabfrage und die eigentliche Spur ginge unter. Wer die SQL-Anweisungen sehen will, stellt den Bereich **„Datenbank (SQL)"** gezielt auf `Debug` .

10.2 Einstellen in der Datei (`appsettings.json`)

Für unbeaufsichtigte Installationen und für Rechner ohne Server (eigener Collector, Arbeitsplatz) steht derselbe Abschnitt in der `appsettings.json` der jeweiligen Anwendung:

```
"Logging": {
  "File": {
    "Enabled": true,
    "Directory": "C:\\ProgramData\\InfraDesk\\Logs",
    "Level": "Information",
    "RollingInterval": "Day",
    "RetainedDays": 31,
    "FileSizeLimitMb": 50,
    "Format": "Text",
    "Components": { "wallboard": { "Enabled": false } },
    "Areas": { "database": "Debug" }
  }
}
```

Einstellung	Bedeutung
`Enabled`	Datei-Protokoll an (`true`) oder aus (`false`). **Vorgabe: an.**
`Directory`	Zielordner. Muss für das Dienstkonto beschreibbar sein. Leer = Vorgabe der Anwendung.
`Level`	`Debug` (sehr ausführlich, für die Fehlersuche) · `Information` (Vorgabe) · `Warning` · `Error`
`RollingInterval`	Wann eine neue Datei beginnt: `Hour` · `Day` (Vorgabe) · `Month` · `Year` · `Infinite` (nie).
`RetainedDays`	Wie viele **Dateistände** aufbewahrt werden (Vorgabe 31). Nicht Tage: Bei Größenumbruch entstehen mehrere Stände pro Tag.
`FileSizeLimitMb`	Ab dieser Größe wird auch innerhalb des Takts eine neue Datei begonnen.
`Format`	`Text` (lesbar, Vorgabe) oder `Json` – eine JSON-Zeile je Ereignis im Format **CLEF**.
`Components`	Ein/Aus (und optional eine eigene Stufe) je Anwendung. Schlüssel: `api` , `portal` , `worker` , `client` , `wallboard` , `clientupdate` , `updater` , `updateagent` , `mobile` .
`Areas`	Mindeststufe je Bereich, z. B. `database` , `http` , `ticketing` , `identity` .

Rangfolge: Was in der Oberfläche gespeichert wurde (`operations.json`), liegt **über** der `appsettings.json` . Die Datei ist also die Auslieferungsvorgabe, nicht das letzte Wort.

Eine Zeile sieht so aus – Zeitpunkt, Stufe, **Quelle**, Meldung und der Zusatzkontext:

```
2026-08-08 09:12:03.118 +02:00 [INF] [InfraDesk.Modules.Ticketing.TicketService] Vorgang T-1042 ges
{"CorrelationId":"7f3c…","TenantId":"…","User":"m.schulz","Component":"api","AppVersion":"2026.08.0
```

Die **Korrelations-ID** ist dabei der Faden durch die Anlage: Client, Portal und App schicken sie mit, die Schnittstelle übernimmt sie und gibt sie zurück. Dieselbe Zeichenfolge steht danach in `client-JJJJMMTT.log` **und** in `api-JJJJMMTT.log` – eine Meldung vom Arbeitsplatz lässt sich damit ohne Raten der passenden Server-Anfrage zuordnen.

10.2a Format `Json` (CLEF) für SIEM und Log-Agenten

Stellen Sie das Format auf `Json` , schreibt jede Anwendung **eine JSON-Zeile je Ereignis**. Das lesen Fluent Bit, Filebeat, Vector, Seq, Elastic und Loki ohne eigene Zerlegungsregel ein, und jedes Zusatzfeld (Korrelations-ID,

Mandant, Benutzer, Version, Rechner) bleibt ein eigenes Feld statt in einer Textzeile zu verschwinden. Für das Lesen von Hand ist `Text` besser geeignet.

10.3 Zentrale Sammlung (mehrere Server, SIEM)

Verteilt Ordivis Platform sich über mehrere Server (API und Portal getrennt), stellt sich die Frage nach einer zentralen Auswertung. Dafür gilt ein Grundsatz:

> **Schreiben Sie die Logs nicht direkt auf eine Netzwerkfreigabe.** Das ist naheliegend und trotzdem falsch: Ein Log auf einem SMB-Share fällt **genau dann aus, wenn man es braucht** – bei einer Netzwerkstörung. Es kann die Anwendung blockieren (ein Schreibzugriff über SMB kann sekundenlang hängen), es erfordert Schreibrechte des Dienstkontos auf einer Freigabe, und mehrere Server kollidieren in derselben Datei.

Der übliche Weg trennt **Schreiben** und **Einsammeln** in zwei Schritte. Das Einsammeln darf ausfallen, ohne dass Ordivis Platform davon berührt wird:

Verfahren	Wann sinnvoll
Windows Event Forwarding (WEF)	Bordmittel, kein Zusatzserver. Die Dienste schreiben ohnehin ins Ereignisprotokoll; WEF sammelt es auf einem Collector ein. Der einfachste Einstieg.
Log-Agent (Fluent Bit, Filebeat, Winlogbeat, Promtail …)	Liest die Dateien aus `Logs\` und liefert sie an Elastic, Loki, Graylog, Seq oder Splunk. Der Standardweg, wenn eine Log-Plattform vorhanden ist.
Syslog / SIEM	In Behörden verbreitet und für ein ISMS oft ohnehin gefordert. Ein Syslog-Agent liest dieselben Dateien.
Geplante Kopie auf eine Freigabe	Für kleine Installationen ausreichend: ein Task, der die Logs nachts wegsichert. **Kopieren statt hineinschreiben** – die Anwendung merkt nichts davon.

Ordivis Platform schreibt bewusst **nur lokal** und macht keine Vorgaben darüber, womit Sie einsammeln.

10.4 Weitere Protokolle

- **Update-Läufe:** `C:\ProgramData\InfraDesk\update-logs\update_<version>_<zeit>.log` – jeder Schritt eines Update-Laufs mit Zeitstempel, **eine Datei je Lauf**. **Bei jedem Update-Problem als Erstes hier hineinsehen**: Die Meldung im Client ist knapp, dieses Protokoll nennt den tatsächlichen Abbruchgrund. Dieselben Zeilen stehen zusätzlich in `updater-JJJJMMTT.log` und unterliegen dort Ihrer Einstellung.
- **Ereignisprotokoll (Windows):** Die Dienste schreiben zusätzlich dorthin – nützlich, wenn ein Dienst gar nicht erst startet und deshalb keine Datei anlegen konnte.
- **Startprotokolle:** `client-error.log` (Arbeitsplatz) und `bootstrapper_<zeit>.log` (Update) halten fest, was **vor** dem Hochfahren des regulären Protokolls passiert ist. Sie unterliegen daher weder Stufe noch Rotation – dafür gibt es sie auch dann, wenn eine Anwendung gar nicht erst so weit kommt.
- **Audit-Log:** revisionssichere Fachaktionen, einsehbar im Client (ID-27). Das ist ein Fach-, kein Technikprotokoll.

10.5 Support-Paket – ein Klick statt zehn Rückfragen

Wenn Sie beim Hersteller-Support anrufen, braucht dieser als Erstes den **Zustand Ihrer Anlage**. Statt ihn über mehrere Rückfragen zusammenzutragen, erzeugen Sie ihn in einem Schritt:

Einstellungen → Protokolle (Logs) → „Support-Paket erstellen"

Das Paket enthält:

Teil	Inhalt
`00-STECKBRIEF.txt`	Lizenznehmer und Produkt, alle Mandanten mit Status, installierte Versionen je Anwendung, Betriebssystem, Laufzeitumgebung, Rechnername, Startzeit des Dienstes, Zeitzone – und Ihre Schilderung des Vorfalls.
`01-INHALT.txt`	Was aufgenommen wurde **und was nicht, mit Grund**.
`logs/`	Die Protokolldateien aller Anwendungen der gewählten Anzahl Tage (Vorgabe 3).
`update-logs/`	Die Protokolle der Update-Läufe samt Zustand des letzten Laufs.
`config/`	Die Konfigurationsdateien – **Kennwörter, Token und Schlüssel geschwärzt**.
`eventlog.txt`	Die Ordivis-Einträge des Windows-Anwendungsprotokolls.

Was NICHT enthalten ist: keine Fachdaten (keine Vorgänge, Objekte oder Benutzerkonten), keine Datenbankinhalte und keine Geheimnisse. Das Paket beschreibt den *Betriebszustand*, nicht den Inhalt Ihrer Anlage – es lässt sich also ohne lange Abwägung verschicken.

Das Archiv ist ein **7z-Archiv mit AES-256**, einschließlich der Dateinamen: Ohne Kennwort lässt sich nicht einmal auflisten, was darin liegt.

Das Kennwort steht bewusst NICHT in der Mail. Ein verschlüsseltes Archiv, dessen Kennwort danebensteht, ist gegenüber jedem, der die Mail sieht, nicht verschlüsselt. Sie bekommen es im Dialog angezeigt (fünf Vierergruppen, ohne verwechselbare Zeichen wie `0` / `O` – es lässt sich vorlesen) und geben es dem Support auf dem Weg weiter, auf dem Sie ohnehin gerade sind: im laufenden Telefonat.

Versendet wird an **support@ordivis.eu**, über das E-Mail-Postfach Ihrer Organisation – der Support kann also antworten. Zwei Fälle, in denen nicht versendet wird und das Paket stattdessen zum **Herunterladen** bereitsteht:

- Sie haben den Versand abgewählt.
- Das Paket ist größer als die Grenze für E-Mail-Anhänge (10 MB). Dann sagt der Dialog das ausdrücklich – wählen Sie weniger Tage oder übermitteln Sie die Datei auf anderem Weg.

Jede Erstellung wird im Windows-Sicherheitsprotokoll vermerkt (wer, wann, welche Datei, an wen) – das Kennwort selbstverständlich nicht.

Abgrenzung zum Fehlerbericht. Das Käfer-Symbol in der Statusleiste (Kapitel 11) meldet *einen* Vorfall aus Sicht eines Anwenders: Beschreibung, Bildschirmfoto, das Tagesprotokoll seines Arbeitsplatzes. Das Support-Paket ist das Werkzeug des Administrators und beschreibt die ganze Anlage. Beides setzt eine Handlung voraus; es geht nichts im Hintergrund hinaus.

Verwandte Themen

- **ID-02 – Systemvoraussetzungen & Architektur** – Voraussetzungen und Architektur vorab
- **ID-51 – Datenbank-Leistung & Dimensionierung** – Dimensionierung und Datenbank-Leistung
- **ID-40 – Administrationshandbuch** – Konfiguration in der Anwendung
- **ID-30 – Modul Notfall- & Wiederanlaufplanung (DR)** – Wiederanlauf nach einem Ausfall

- **ID-28 – Modul Informationssicherheit (ISMS)** – Härtung und Nachweise