

Zbix CRM Architecture Reference

Product release 1.2.0

Zbix Technologies
Documentation edition 1.2.0-r1
28 October 2026

Application components, data relationships, deployment topologies and managed SaaS readiness.

Use the contents to find a task, then review its prerequisites and expected result. Screenshots show demonstration data and preserve the original interface. Your permissions, organization layouts and enabled capabilities can change the controls you see.

Private review edition. Source-reviewed procedures have not been executed against a CRM installation as part of this documentation project. Platform acceptance and provider configuration require their stated qualification.

Contents

Application architecture overview	3
Technology stack and dependency boundaries.....	6
Frontend architecture and navigation	7
Backend architecture and application services.....	7
Database architecture and transaction integrity.....	9
CRM and commercial data relationships	11
Private files and generated document storage	12
Background processing and durable jobs.....	13
API validation and safe concurrency.....	15
Assess managed SaaS capabilities	17

Application architecture overview

Zbix CRM 1.2.0 | Source reviewed | Reviewed 28 October 2026

Zbix CRM is a modular application with a browser interface, an HTTP API, a separate background worker, PostgreSQL and private file storage. The API and worker share the same domain models and durable state. The browser reaches application services through the web gateway. Optional SMTP and Microsoft Graph connections extend communications; the core CRM does not require a connected Microsoft tenant.

Use this reference to identify the component responsible for a request or delayed operation and to plan an operational handover without assuming additional infrastructure.

Before you begin

- Access to the release manifest and an administrator account for System Health.
- A deployment-specific topology showing the gateway and persistent stores.

Procedure

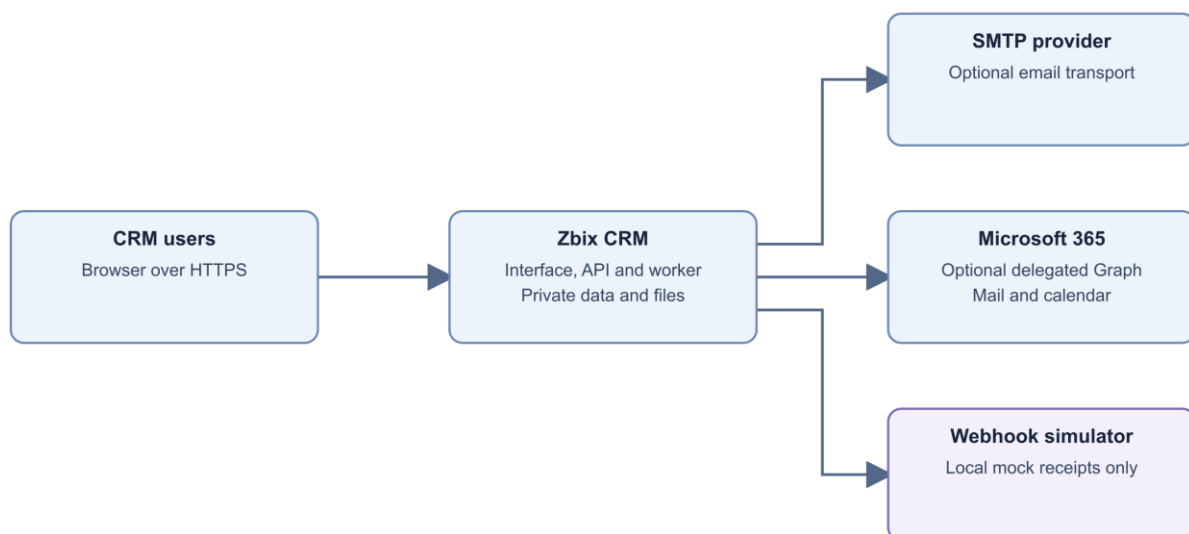
1. Open Administration > System Health and record the displayed migration and available runtime identity values in the private handover record.
2. Identify the browser-to-gateway route, then the gateway-to-API route. Ordinary CRM pages and API calls share the application origin.
3. Identify PostgreSQL and the private file store used by both API and worker. Record their protected backup scope through the approved recovery process.
4. Inspect worker heartbeat and queue summaries separately from database reachability. A working page proves neither scheduled execution nor external delivery.
5. For an integration issue, follow the provider selected for that operation. SMTP, Graph and local mock delivery have different outcomes and prerequisites.

Expected result

The support team can trace browser requests, transactions, queued work and stored documents to their responsible component.

System context

Core CRM and explicitly configured providers

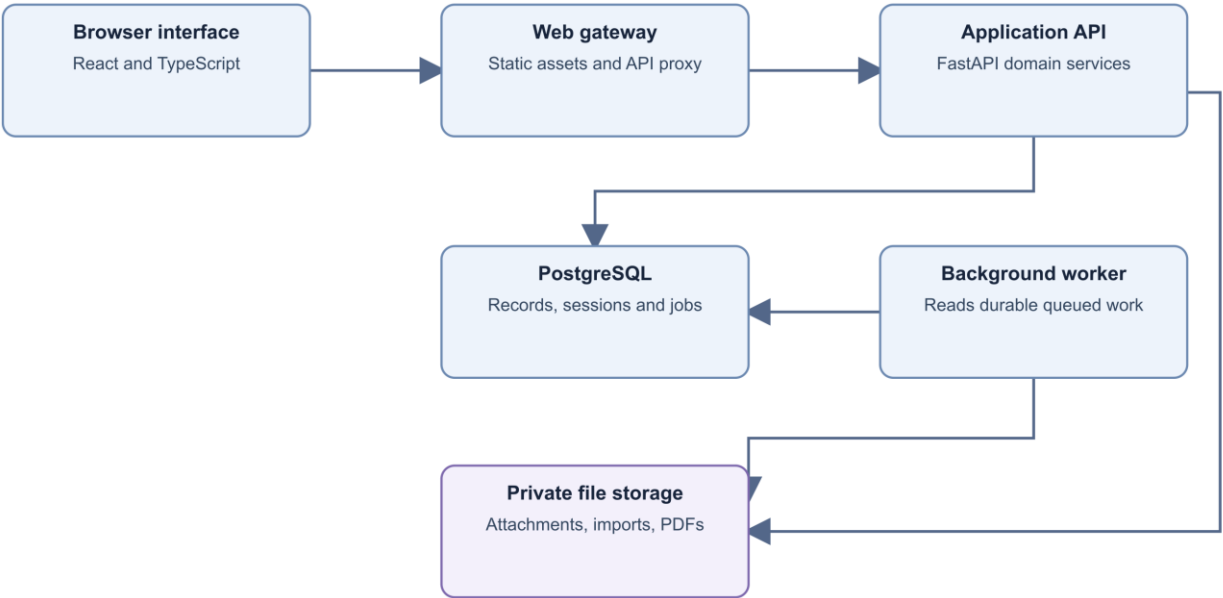


Provider configuration and successful delivery are separate checks.
Microsoft connection does not replace CRM sign in.

Figure 1 System context

Logical application components

Request handling, durable work and private persistence

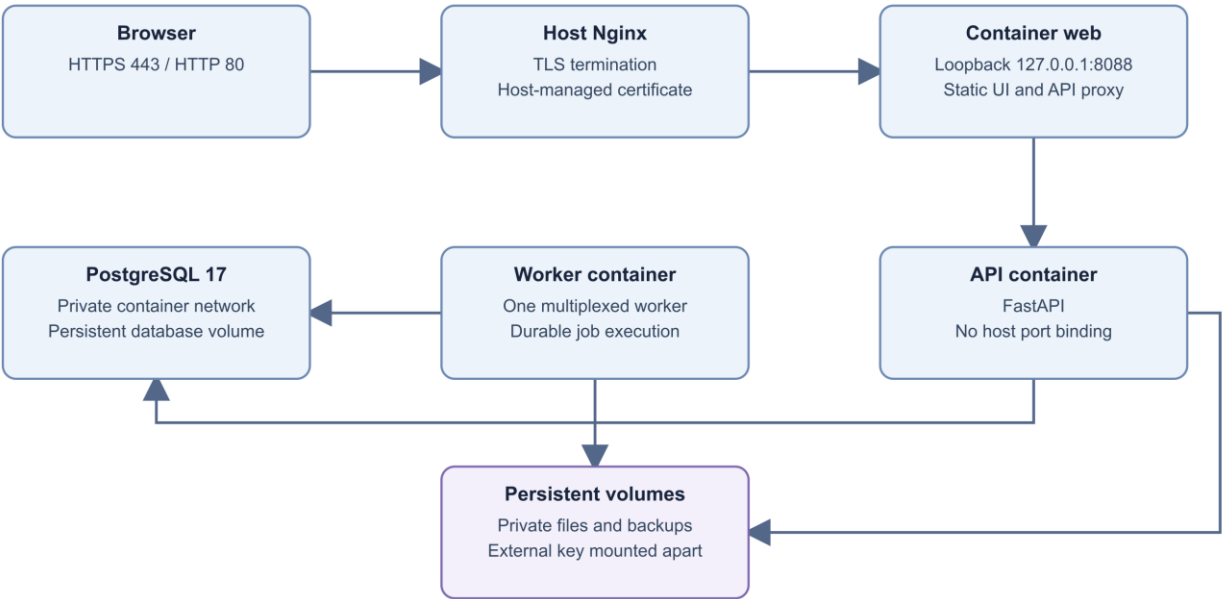


API and worker share models and persistent state.
Arrows show use or dependency; no separate message broker is required.

Figure 2 Logical application components

Linux deployment topology

Source-qualified Ubuntu amd64 package; one host

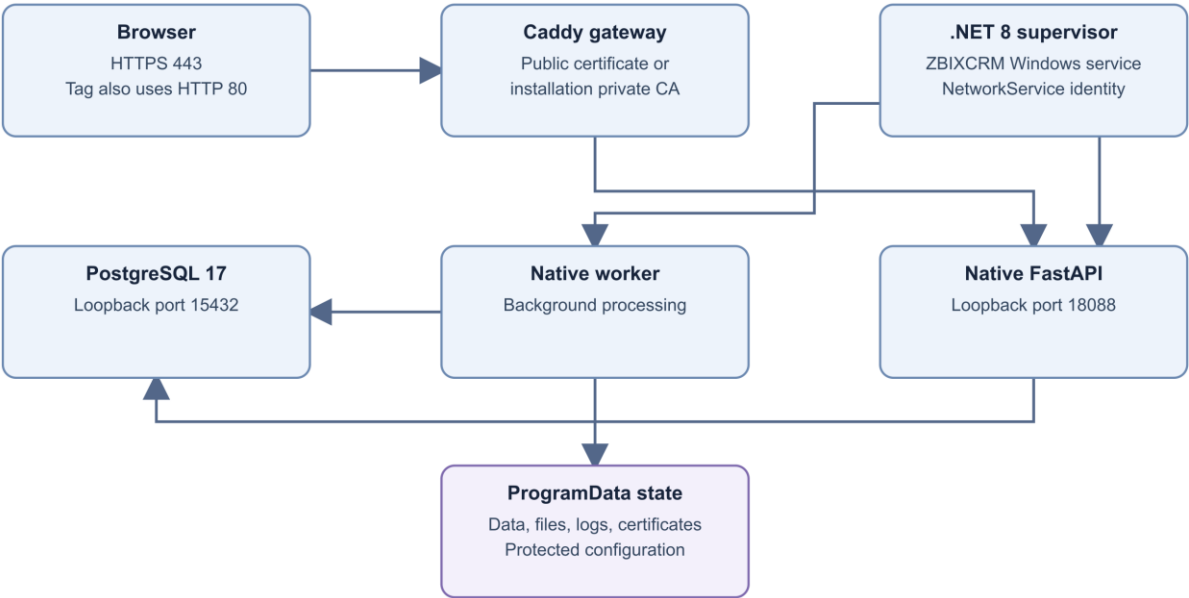


systemd wraps Docker Compose; database and API remain private.
Optional provider egress is configured separately. No HA or external broker is established.

Figure 3 Linux deployment topology

Native Windows topology

Tagged v1.2 runtime; supervised native processes

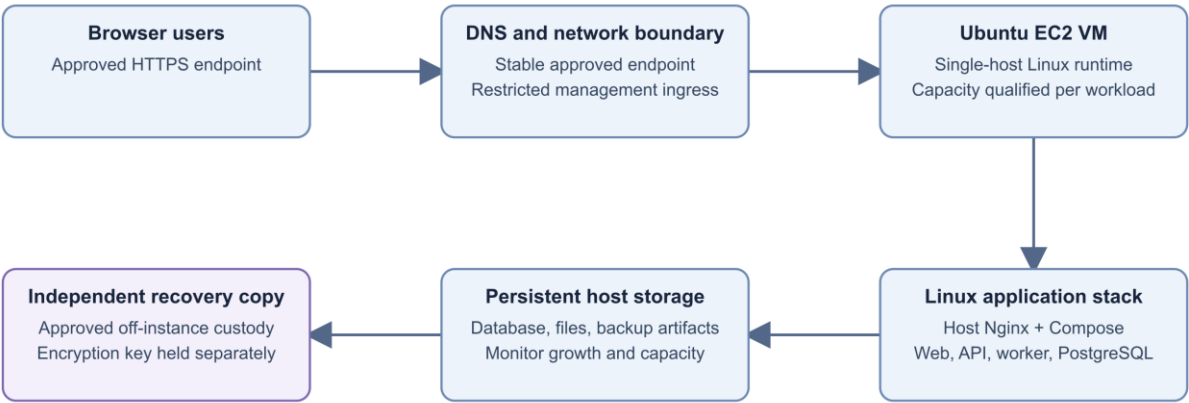


Supervisor also starts PostgreSQL and Caddy; its links illustrate process supervision.
Optional port 80 / IIS coexistence correction is post-tag, not qualified by v1.2 source.

Figure 4 Native Windows topology

AWS EC2 deployment context

Recorded Linux deployment uses the same self-hosted package



No managed load balancer, RDS, HA cluster or managed SaaS is established by this topology.
Off-instance copy and key custody require deployment-specific operational validation.

Figure 5 AWS EC2 deployment context

Important considerations

- No Redis, RabbitMQ or external search engine is required by this implementation. PostgreSQL stores job state and execution coordination.

- The gateway differs by deployment package. Use the Linux or Windows topology for concrete service and port details.
- A source tag identifies code. A screenshot or health response is separate evidence of the running installation. Unreported runtime identity must not be treated as a confirmed match.

Troubleshooting

- Pages load but jobs do not complete: inspect the worker and relevant queue, then the current actor permissions.
- Files are unavailable despite visible records: verify private storage and attachment scan evidence rather than assuming a database failure.

Related knowledge base articles: Technology stack and dependency boundaries; Background processing and durable jobs; Read System Health and verify service readiness; Private files and generated document storage.

Technology stack and dependency boundaries

Zbix CRM 1.2.0 | Source reviewed | Reviewed 28 October 2026

The v1.2.0 source uses React and TypeScript for the interface and FastAPI with Pydantic for the API. SQLAlchemy manages persistence and Alembic manages schema revisions. PostgreSQL provides the release database. ReportLab generates commercial PDFs, while PDF.js displays generated PDF bytes in the browser. The application packages these libraries with its own code; customers should upgrade through a qualified product release.

Use the stack inventory for integration planning, support ownership and dependency review. It does not prescribe installing current upstream library versions into a released appliance.

Before you begin

- The exact product release or installer manifest.
- Access to the release-specific dependency declarations when performing engineering maintenance.

Procedure

1. Identify the product release from its release artifact and migration evidence. Do not use the frontend package version or API title alone as the product version.
2. Review the frontend declaration for React, React DOM, Vite, TypeScript, Lucide icons, local fonts, QR rendering and PDF.js.
3. Review backend declarations for FastAPI, Uvicorn, SQLAlchemy, Alembic, psycopg, Argon2, cryptography, PyOTP, MSAL and the document/file libraries.
4. Use the package build manifest or lock evidence to determine resolved versions. A dependency range is not the installed version.
5. Evaluate changes in an isolated release build, including database compatibility, document output and browser behavior, before replacing customer binaries.

Expected result

The inventory identifies each major technology and the difference between source requirements and a frozen distribution.

Important considerations

- The frontend package still declares 0.6.0 and the FastAPI application metadata declares 0.5.0 in the v1.2.0 tree. These are historical component metadata, not release authority.
- SQLite fallback exists in database initialization, but it does not establish support for PostgreSQL-dependent worker and concurrency behavior in a customer deployment.
- Local fonts avoid making the normal interface dependent on a public font service.

Troubleshooting

- Version labels disagree: compare the release tag, artifact manifest and database migration before selecting documentation.
- A dependency change affects PDFs or imports: retest those features with representative, non-sensitive documents.

Related knowledge base articles: Frontend architecture and navigation; Backend architecture and application services; Database architecture and transaction integrity.

Frontend architecture and navigation

Zbix CRM 1.2.0 | Source reviewed | Reviewed 28 October 2026

The React interface is a single-page application. Its shared shell contains grouped navigation, global search, quick creation and session controls. Hash navigation selects modules and record details. Individual components implement commercial documents, reports, scheduling, automation and administration. Available navigation depends on effective permissions and enabled capabilities; hiding a menu is not the security enforcement mechanism.

Use this article to understand page loading, navigation and client-side behavior when supporting a user or extending a compatible integration.

Before you begin

- An authenticated browser session.
- The user role and module involved in the reported issue.

Procedure

1. Reproduce the problem from the normal sidebar or a record link, and record the visible page title and action. Avoid copying a URL containing a password recovery token.
2. Check whether the module is available for the user. Compare the effective module permissions and capability state with an administrator.
3. Use the page Refresh action after a permission, status or record change. The client briefly caches reads and invalidates its cache after writes.
4. For a validation failure, read the form-level message and the affected field. The API response supplies the authoritative rejection.
5. If a problem appears only at a narrow viewport, reproduce it at desktop width and report both viewport sizes and the affected control.

Expected result

A support report distinguishes unavailable navigation, stale client state, validation feedback and a reproducible layout problem.

Important considerations

- The API client includes credentials and the CSRF header for authenticated operations. Browser storage should never be used to distribute service credentials.
- Record views and layout metadata customize presentation, but server-side permission and field checks remain authoritative.
- The sidebar places Pipelines in Sales, while the breadcrumb helper labels it Automation in this release. Use the visible Pipelines entry and record the labeling inconsistency.

Troubleshooting

- An operation fails after another user edits the record: refresh and reapply the intended change to the latest version.
- A deep link opens an unexpected screen: navigate through the sidebar and compare the release baseline before reporting a broken route.

Related knowledge base articles: Authorization and permission evaluation; API validation and safe concurrency; CRM and commercial data relationships.

Backend architecture and application services

Zbix CRM 1.2.0 | Source reviewed | Reviewed 28 October 2026

FastAPI exposes authenticated CRM operations and specialized routers for commercial documents, workflows, approvals, revenue, integrations, files and administration. Pydantic and domain validation reject unsupported input. SQLAlchemy transactions combine business changes with their audit and scheduling effects. The backend is organized by functional services inside one application codebase; the worker imports those services for delayed processing.

Use the service boundaries to locate the correct support evidence and to avoid treating every visible page as an independent deployment service.

Before you begin

- A known product release.
- A reproducible symptom, the affected module and the approximate time.

Procedure

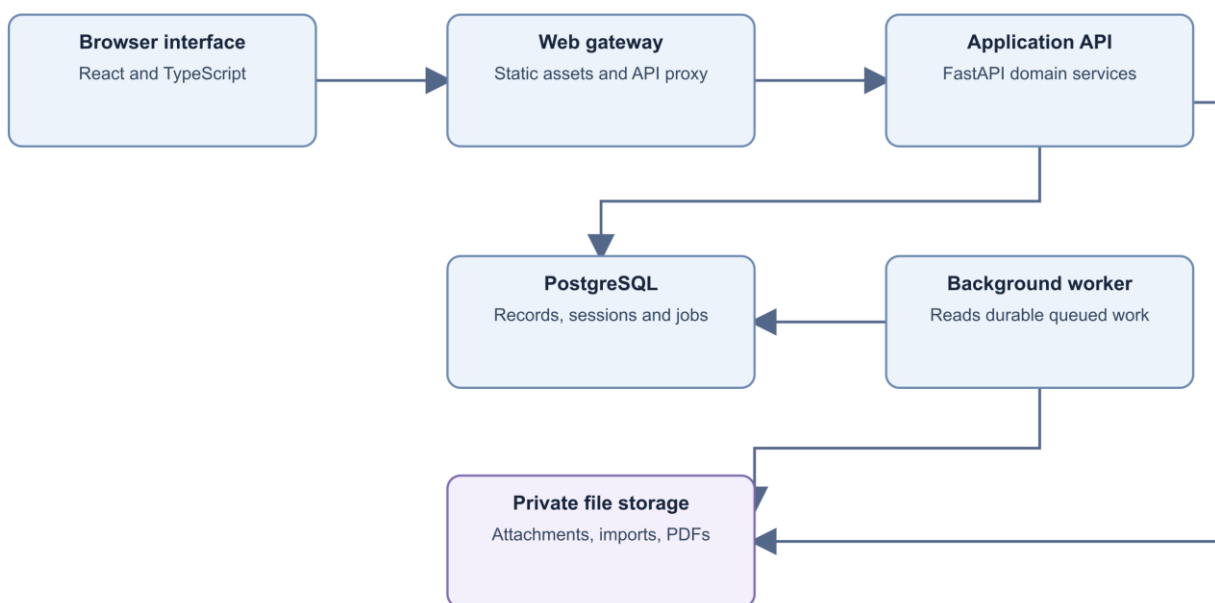
1. Identify whether the failure is an HTTP request or delayed work. Requests are handled by the API; queued work is handled by the worker.
2. Check the response status and safe user-facing message. Use field-level validation feedback for rejected data, and refresh for a stale-version conflict.
3. Check System Health for database, storage and worker evidence before changing application data.
4. For a commercial or automation operation, inspect its history or execution receipt. This identifies which business transition actually committed.
5. Escalate persistent failures with the module, operation, time, request identifier when available and a sanitized reproduction.

Expected result

The incident is associated with the API, a domain rule, a persisted job or an infrastructure dependency, making corrective work specific.

Logical application components

Request handling, durable work and private persistence



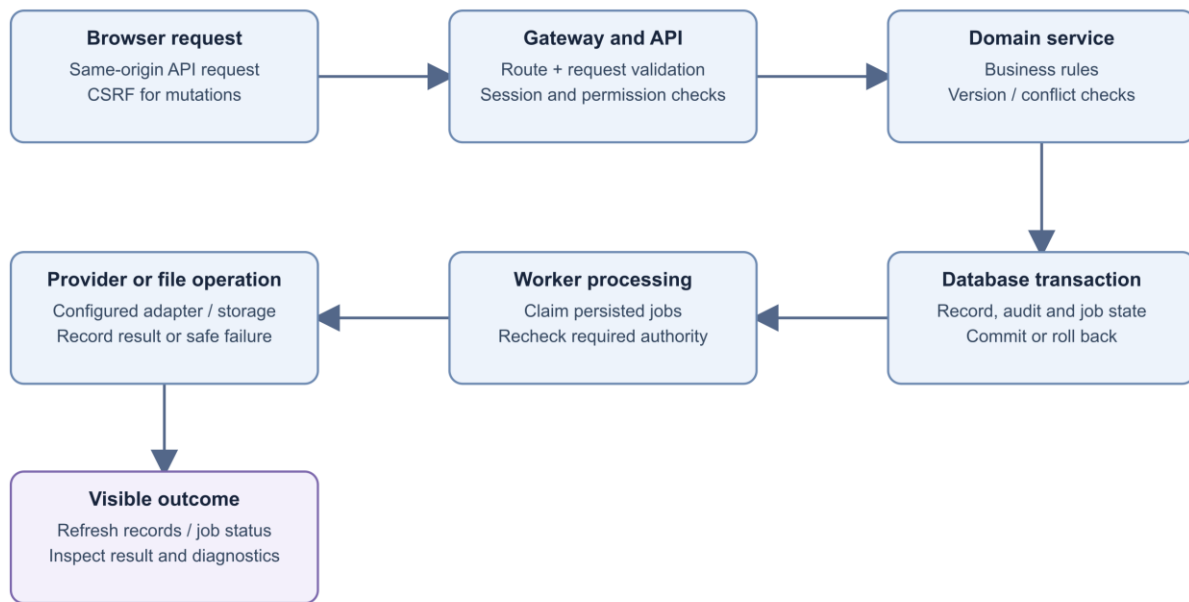
API and worker share models and persistent state.

Arrows show use or dependency; no separate message broker is required.

Figure 6 Logical application components

Request and data flow

Validated transactions and independently executed background work



Synchronous requests return after their own transaction; queued execution occurs later.
An accepted request or queued state does not establish external delivery.

Figure 7 Request and data flow

Important considerations

- Authentication, CSRF, authorization and licensed capability checks run on the server. Direct API requests do not bypass them.
- Core record updates use optimistic versions; conversions and commercial transitions also use transactional locking and uniqueness constraints.
- The safe server entry point is app.server. Ad hoc server commands can bypass the intended logging configuration and should not replace package service definitions.

Troubleshooting

- A conflict response follows repeated submission: inspect the current record and operation history before submitting again.
- An unexpected error persists: preserve the safe diagnostic context and contact support; do not repair live records directly in the database.

Related knowledge base articles: API validation and safe concurrency; Database architecture and transaction integrity; Background processing and durable jobs; Operational logging and support evidence.

Database architecture and transaction integrity

Zbix CRM 1.2.0 | Source reviewed | Reviewed 28 October 2026

PostgreSQL stores organization identities, users, sessions, CRM records, permissions, commercial documents and durable automation state. Seven core modules share a records table with typed common columns and module-specific JSON. Dedicated tables hold commercial snapshots, custom values, templates, scheduling, approval processes and revenue agreements. This is a relational application with selected JSON structures, not a document-only database.

Use this reference to understand data ownership, migration boundaries and why supported application operations are necessary for consistent changes.

Before you begin

- Authorized database administration responsibility.
- Release-specific migration and recovery evidence.

Procedure

1. Confirm the running migration using System Health runtime identity or the installation runbook. Compare it with the intended release before maintenance.
2. Identify the business object and its related records through the application. Common CRM links include account, contact, deal, parent and related record.
3. Treat document snapshots, history and scheduled work as part of the object lifecycle. Updating a single database value can leave those records inconsistent.
4. Before a release change, use a verified coordinated recovery point and rehearse the forward migration on isolated targets.
5. After authorized maintenance, compare migration state and health with the approved plan and perform representative read-only business checks.

Expected result

Database maintenance preserves the relationship graph, commercial history and durable work expected by the application.

Core and commercial record relationships

Conceptual links and controlled conversions; not a physical ER schema



Arrows show supported associations or conversion direction, not universal one-to-one cardinality.
Organization boundaries and selected parent-record permissions constrain access.

Figure 8 Core and commercial record relationships

Important considerations

- Composite organization/reference constraints prevent cross-organization relationships for modeled links; service checks also validate module type and business consistency.
- Money uses fixed-precision values with explicit currencies. Do not add different currencies without a documented conversion rule.
- JSON fields and application validation are part of the schema contract. The presence of a database column does not make direct editing a supported workflow.
- A database dump alone omits private file bytes and external recovery keys. Follow the coordinated recovery guide.

Troubleshooting

- A relationship update is rejected: check organization, module, visibility and account/contact consistency.
- Migration and application versions differ: stop the change and reconcile the release plan before writing data.

Related knowledge base articles: CRM and commercial data relationships; Private files and generated document storage; Backend architecture and application services.

CRM and commercial data relationships

Zbix CRM 1.2.0 | Source reviewed | Reviewed 28 October 2026

Accounts represent organizations; Contacts represent people and may refer to Accounts. Deals may link an Account and primary Contact. Quotes may link those records and a Deal, with currency and relationship consistency enforced. A qualified Lead can be converted into a new or existing Account and Contact, with an optional Deal. Conversion records the resulting identities and preserves the original Lead as Converted.

Use the relationship model to prepare clean data, resolve lookup errors and understand the path from qualification to invoicing.

Before you begin

- View access to related records and the appropriate create or edit permissions.
- A qualified Lead for conversion, or a valid sales document source for commercial conversion.

Procedure

1. Search for an existing Account before creating another organization record. When converting a Lead, select the existing Account if a matching company already exists.
2. Select a Contact associated with the chosen Account. If linking an existing Deal during conversion, its Account and primary Contact must match and the Deal must remain open.
3. Keep Quote currency and relationships consistent with the linked Deal. Review the current accepted quotation revision before converting it to a Sales Order.
4. Confirm a Sales Order before creating its Invoice. The Invoice inherits its commercial snapshot rather than recalculating prices from the current product catalog.
5. Use the associated history and revision selector when reviewing prior terms. Preserve source references instead of recreating historical documents as ordinary records.

Expected result

Related records form a consistent customer and sales history, and commercial documents retain the terms approved at the relevant stage.

Core and commercial record relationships

Conceptual links and controlled conversions; not a physical ER schema



Arrows show supported associations or conversion direction, not universal one-to-one cardinality.
Organization boundaries and selected parent-record permissions constrain access.

Figure 9 Core and commercial record relationships

Important considerations

- Converted Leads cannot be edited through ordinary record updates. Conversion retries return the stored conversion result.
- Existing records selected during Lead conversion are linked; conversion does not silently replace their field values.
- An order from an accepted quotation snapshot is financially immutable. Amend through a new quotation revision.
- A source Sales Order has one Invoice relationship in this implementation, including when that invoice is later voided.

Troubleshooting

- A Contact lookup or save fails: verify that it belongs to the selected Account and is visible to the current user.
- Quote conversion is unavailable: ensure the selected revision is the current Accepted revision and that Sales Order create permission is granted.

Related knowledge base articles: Database architecture and transaction integrity; Authorization and permission evaluation; API validation and safe concurrency.

Private files and generated document storage

Zbix CRM 1.2.0 | Source reviewed | Reviewed 28 October 2026

CRM attachment metadata is stored in the database while attachment bytes live in a private durable file store. Objects use generated opaque keys rather than user-supplied paths. Background import sources occupy a separate namespace in the same private storage root. Commercial document finalization captures document snapshots and generated artifacts so later catalog or template edits do not silently alter retained documents.

Use this article when planning storage, investigating missing files or defining the scope of a recovery point.

Before you begin

- Administrator access to System Health and the package storage configuration.
- A known attachment or document reference; do not distribute the underlying private filesystem path.

Procedure

1. Check System Health for private storage availability, writability and remaining capacity. Confirm the API and worker use the intended durable store.
2. Open the parent record and its Attachments tab. Verify that the current user can access both the record and the attachment action.
3. Review the scan state before requesting a download. A listed filename is not proof that the bytes are released for download.
4. For an import, inspect Import Jobs and its source/progress history rather than treating the source file as an ordinary customer attachment.
5. For a finalized sales document, use the document PDF action and selected revision. Compare the document identity and revision when investigating a historical rendering difference.

Expected result

The investigation separates metadata, private bytes, scan evidence and stored commercial output.

Important considerations

- Supported attachment types are PDF, DOCX, XLSX, PNG, JPG/JPEG, TXT and CSV, subject to content validation. Upload limits default to 10 MiB with a hard cap of 50 MiB; the installed proxy may impose a lower effective limit.
- Removing an attachment from a record does not necessarily purge its bytes immediately. Retention cleanup is a worker operation.
- Import sources and completed import histories have no general automatic retention cleanup in the reviewed implementation.
- Protect the entire private store as part of coordinated recovery. Do not expose its directory through the web gateway.

Troubleshooting

- File metadata appears but download fails: inspect scan status and parent permissions, then storage integrity.

- Disk usage grows after imports: review retained import sources and establish an approved retention process rather than deleting files ad hoc.

Related knowledge base articles: Troubleshoot attachment upload and download; Troubleshoot import validation and execution; Read System Health and verify service readiness.

Background processing and durable jobs

Zbix CRM 1.2.0 | Source reviewed | Reviewed 28 October 2026

Zbix CRM runs a separate worker process for scheduled actions, background imports, email, calendar writes, mailbox synchronization, mock webhooks, managed backup work and attachment scanning. PostgreSQL stores job state and execution coordination. The worker performs bounded work in successive cycles and publishes a persisted heartbeat. A browser request that queues work does not mean the work has finished.

Use this article to follow scheduled operations and to distinguish a queue delay from a completed or failed business operation.

Before you begin

- Administrator access to System Health or permission to view the relevant operation history.
- The job, import, communication or automation record involved.

Procedure

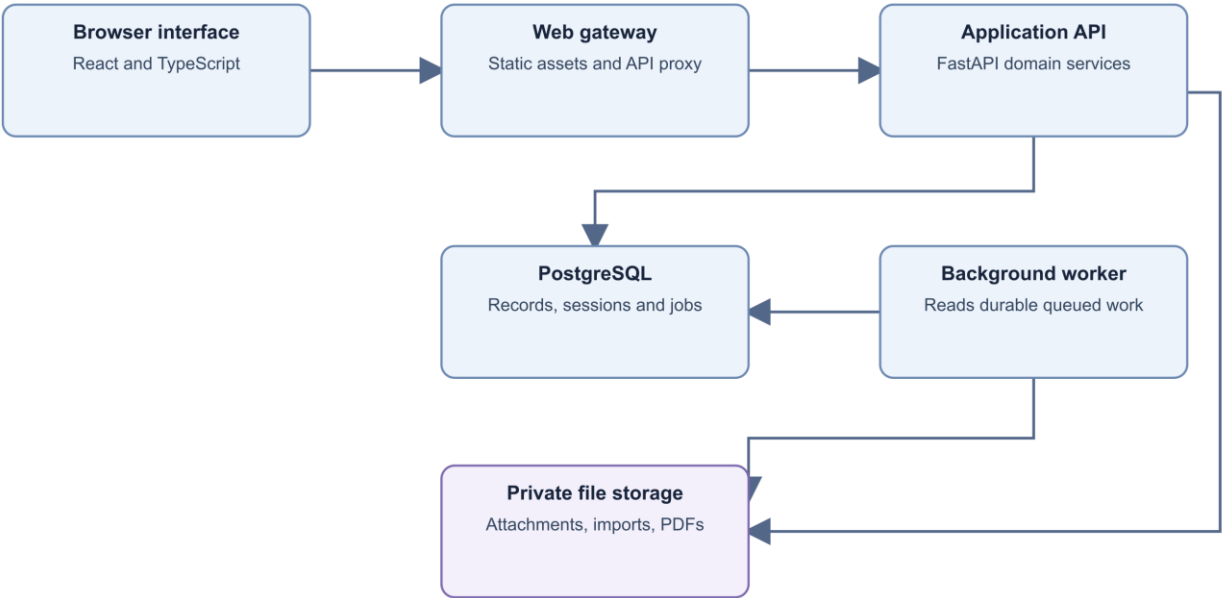
1. Open the operation record and note its state, scheduled time, latest attempt and safe error. Confirm whether the operation is due.
2. Open System Health and compare worker heartbeat age with the queue counts and oldest due-work age. A healthy heartbeat is recent and reports running.
3. Check the initiating user is active and still has the permissions and related-record visibility required by the operation. Workers re-evaluate authority.
4. Use the operation-specific cancel or retry control only after reviewing its semantics. Import cancellation preserves completed batches; uncertain external sends require provider-side review.
5. After resolution, refresh the record and verify a persisted terminal outcome or execution receipt, not only an empty queue count.

Expected result

The operator can explain whether work is waiting, retrying, completed, cancelled or failed, and verify the effect independently of the initiating request.

Logical application components

Request handling, durable work and private persistence

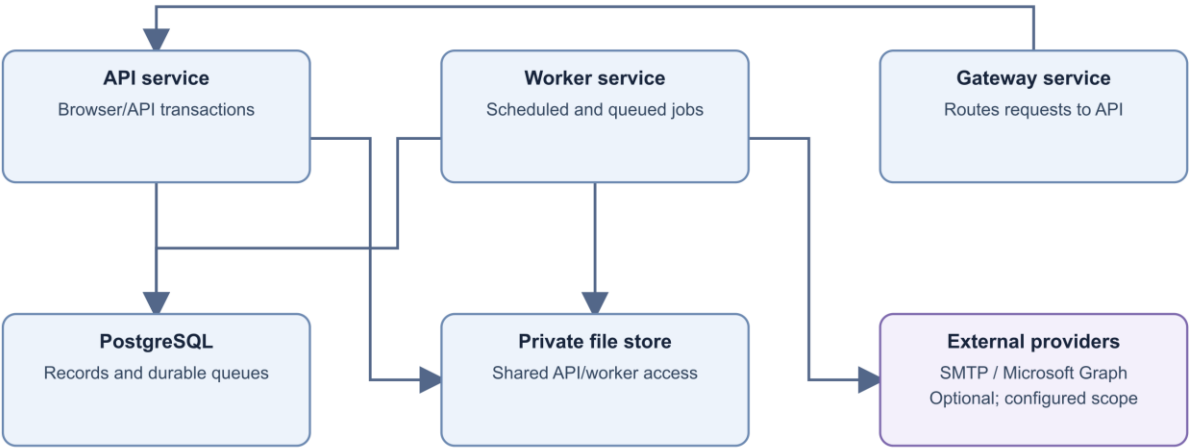


API and worker share models and persistent state.
Arrows show use or dependency; no separate message broker is required.

Figure 10 Logical application components

Service dependencies

Dependencies to check when only part of the application is healthy

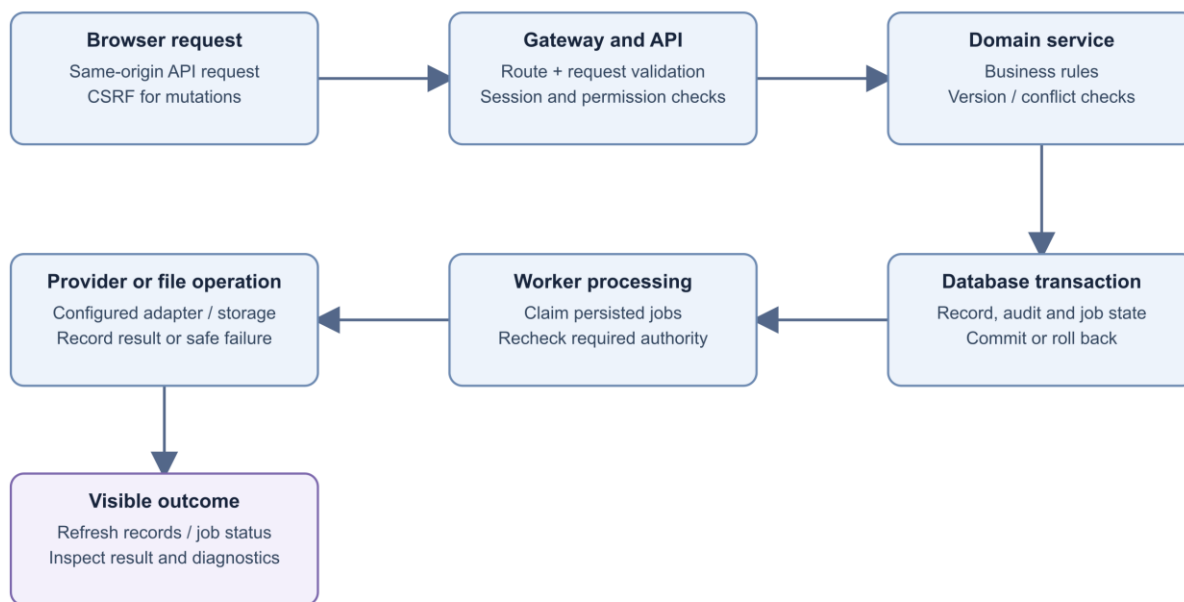


Database readiness does not prove worker progress or provider delivery.
Provider availability depends on configured operation and actor authorization.

Figure 11 Service dependencies

Request and data flow

Validated transactions and independently executed background work



Synchronous requests return after their own transaction; queued execution occurs later.
An accepted request or queued state does not establish external delivery.

Figure 12 Request and data flow

Important considerations

- The worker usually waits two seconds when idle and retries its outer cycle after five seconds on a cycle failure. These are implementation intervals, not delivery-time guarantees.
- Hourly maintenance handles deleted/orphaned attachment cleanup and expired backup leases.
- Worker restart is designed around durable state. It does not justify repeatedly submitting an external action whose outcome is uncertain.
- Different job families use their own locking and retry rules; there is no universal retry policy for every queue.

Troubleshooting

- Due jobs accumulate with a stale heartbeat: investigate the worker service and database connection.
- One operation remains blocked while others progress: inspect that operation's policy, actor permissions and execution history.

Related knowledge base articles: Read System Health and verify service readiness; Operational logging and support evidence; Troubleshoot queued failed or uncertain email; Troubleshoot import validation and execution.

API validation and safe concurrency

Zbix CRM 1.2.0 | Source reviewed | Reviewed 28 October 2026

The application validates data both at request boundaries and in domain services. Records carry versions, and mutations can reject stale versions rather than overwrite another user's change. Relationships must refer to accessible records of the expected type in the same organization. Commercial status transitions apply additional rules beyond ordinary field validation.

Use this reference to resolve rejected forms and design clients that preserve user intent when data changes concurrently.

Before you begin

- The current record or operation response and its version.
- Permission for the requested action and accessible related records.

Procedure

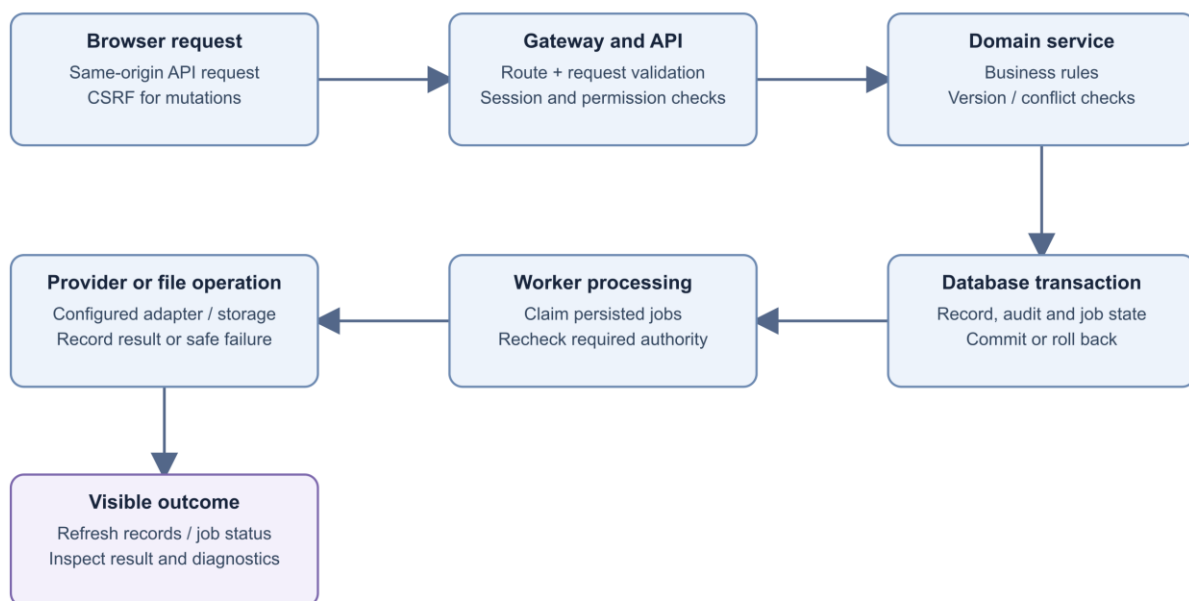
1. Read the latest record before editing and keep its version with the proposed change. Submit only supported fields.
2. Validate names, email addresses, dates, amounts and required stage fields before submission. Treat the server's detailed field response as authoritative.
3. If the server reports that the record changed, reload it, compare the new state and reapply only the intended changes. Do not repeatedly resend the stale body.
4. For an operation with an idempotency key, retain the key for a retry of the same request. Never reuse it for changed content.
5. After a successful action, inspect the returned state and history. A saved field value does not prove a subsequent worker or provider operation completed.

Expected result

The client handles invalid data, concurrent changes and retries without silently duplicating business effects.

Request and data flow

Validated transactions and independently executed background work



Synchronous requests return after their own transaction; queued execution occurs later.
An accepted request or queued state does not establish external delivery.

Figure 13 Request and data flow

Important considerations

- Core names are required and limited to 200 characters. Email validation is bounded, money must be finite and nonnegative, and currencies use three uppercase letters at the core record layer.
- Lead and Contact email deduplication is case-insensitive within the module and organization; Product codes also have duplicate protection.
- Meeting end time must follow start time. Calendar operations also validate timezone behavior.
- An integration endpoint supports only its published operations; browser endpoints are not automatically available through a bearer credential.

Troubleshooting

- Conflict after a double click: inspect the current operation result before creating another record.
- A relationship fails validation: confirm account/contact/deal consistency and current visibility instead of substituting a hidden record identifier.

Related knowledge base articles: CRM and commercial data relationships; Configure and use the scoped integration API; Backend architecture and application services.

Assess managed SaaS capabilities

Zbix CRM 1.2.0 | Source reviewed | Reviewed 28 October 2026

Zbix CRM includes organization-scoped data and a bounded SaaS entitlement adapter. These foundations are distinct from a complete managed subscription service. A self-hosted CRM running in the cloud does not automatically provide automated tenant onboarding, centralized billing, high availability or certified tenant recovery.

Classify existing capabilities accurately when evaluating a managed-service requirement or preparing an enterprise architecture review.

Before you begin

- An identified managed-service use case and intended tenant-isolation model.
- Access to the release-specific capability evidence and operational requirements.
- Separate owners for entitlement authority, provisioning, billing, monitoring and recovery.

Procedure

1. Identify which responsibilities belong to the CRM application and which would require a managed-service control plane. Record the chosen tenancy and recovery boundaries before onboarding customers.
2. Classify organization-scoped records and authorization as implemented application controls. Validate their behavior for the actual service design; do not infer shared-platform operational isolation from an organization field alone.
3. Classify the SaaS entitlement cache and update adapter as partially integrated. It accepts versioned authority updates, maintains tenant/organization binding and fails licensed writes closed after the configured staleness window.
4. Keep online subscription authority, payment processing and customer billing integration separate. The reviewed central license manager does not provide the production online subscription/revocation service needed to complete that path.
5. Classify provider recovery as a restricted handoff contract. The ordinary worker does not execute managed database snapshots or point-in-time recovery. Dedicated logical-instance assumptions apply; pooled-tenant recovery is not certified.
6. Require separate evidence for automated tenant provisioning, centralized authentication, global monitoring, availability/scaling, tenant-specific recovery and coordinated service upgrades. Mark these as not established by the reviewed release rather than claiming they are ready.
7. Approve a managed-service offer only after its missing implementation and operational qualifications are completed. Keep proposed future architecture separate from current product operating instructions.

Expected result

The architecture review distinguishes implemented CRM controls, partially integrated SaaS support and service capabilities that remain unqualified or not established.

Fields and controls

Field or control	Rule
Organization-scoped CRM data and permissions	Implemented; validate the intended tenancy design separately.
Cached SaaS entitlement adapter	Partially implemented integration; isolated evidence, not a completed production subscription authority.
Online subscriptions and payment billing	Not supported as a completed central integration in the reviewed release.
Provider snapshot and PITR execution	Not supported by the ordinary CRM worker; restricted handoff contracts only.
Pooled-tenant recovery	Not production-qualified.
Managed provisioning, centralized SSO and global control plane	Not established by the reviewed release evidence.

Field or control	Rule
HA, service scaling and managed upgrade orchestration	Not established; require separate architecture and qualification.

Important considerations

- Implemented code is not equivalent to production qualification. The entitlement adapter has isolated integration evidence but no completed online subscription/billing authority in the reviewed release.
- Cached entitlement freshness is bounded; an unreachable authority does not imply unlimited licensed operation.
- A tenant administrator does not gain provider recovery authority merely by administering CRM records.
- No managed-service availability guarantee, security certification or production RPO/RTO is established by this documentation.

Troubleshooting

- If licensed writes fail in SaaS mode, have the service operator review entitlement freshness, version and authority synchronization rather than changing user roles indiscriminately.
- If recovery is queued as a provider handoff, confirm the connected authorized recovery process and do not present it as an executed restore.
- Escalate commercial or architecture claims that depend on unverified control-plane capabilities to the service owner before customer commitment.

Related knowledge base articles: Choose a deployment model; Understand recovery points and backup services; Activate and review a self-hosted license; Plan recovery and rollback decisions.