LetsBe — Repository Analysis

Codebase Audit and Analysis

Version: v1.0
Date: February 26, 2026
Company: LetsBe Solutions LLC
Contact: matt@letsbe.solutions

221 North Broad Street, Suite 3A, Middletown, DE 19709

Contents

1. LetsBe Platform — Comprehensive Architecture Analysis

Generated: 2026-02-25 **Scope:** All 5 repositories in the LetsBe workspace

---

2. Table of Contents

---

3. Repository 1: letsbe-hub

3.1 1. Overview

- **Language/Framework:** TypeScript (strict), Next.js 16.1.1 (App Router, standalone output, Turbopack), React 19.2.3
- **Runtime:** Node.js 20 (Alpine in Docker)
- **Approximate LOC:** ~12,000–15,000 across 244 source files (156 `.ts` + 88 `.tsx`)
- **Key Dependencies:**
  - **ORM:** Prisma 7.0.0 with `@prisma/adapter-pg` (PostgreSQL native driver)
  - **Auth:** NextAuth.js 5.0.0-beta.30 (Credentials provider, JWT sessions)
  - **Payments:** Stripe 17.7.0
  - **SSH:** ssh2 1.17.0
  - **State:** TanStack Query v5, Zustand 5.0.3
  - **UI:** Tailwind CSS 3.4.17, shadcn/ui (Radix primitives), Recharts 3.6.0
  - **Validation:** Zod 3.24.1
  - **HTTP:** undici 7.18.2 (Portainer HTTPS with self-signed certs)
  - **Email:** nodemailer 7.0.12
  - **2FA:** otplib 13.1.0, qrcode 1.5.4
  - **Storage:** @aws-sdk/client-s3 3.968.0 (S3/MinIO)
  - **Testing:** Vitest 4.0.16, @testing-library/react 16.3.1

- **Deployment:**
  - `Dockerfile`: Multi-stage build (deps → builder → runner), Node 20 Alpine, installs `docker-cli`, runs `startup.sh` (Prisma migrate deploy → node server.js)
  - `deploy/docker-compose.yml`: `db` (postgres:16-alpine) + `hub` (image: `code.letsbe.solutions/le` port `127.0.0.1:3847:3000`, mounts Docker socket + jobs/logs dirs
  - `.gitea/workflows/build.yml`: Gitea Actions — lint+typecheck job → Docker build+push to `code.letsbe.solutions/letsbe/hub`
  - `deploy/nginx/hub.conf`: Reverse proxy to `localhost:3847`

3.2 2. Architecture

3.2.1 Entry Points

- **Dev:** `npm run dev` → `next dev` → `http://localhost:3000`
- **Production:** `startup.sh` → `prisma migrate deploy` then `node server.js`
- **DB Seed:** `npm run db:seed` → `tsx prisma/seed.ts`
- **Root page:** `src/app/page.tsx` — redirects to `/admin` or `/login`

3.2.2 Core Modules

Module	Path	Description
`automationWorker`	`src/lib/services/automationWorker.ts`	State machine for order provisioning flow (PAYMENT_CONFIRMED → AWAITING_SERVER → SERVER_READY → DNS_PENDING → DNS_READY → PROVISIONING → FULFILLED). Handles AUTO/MANUAL/PAUSED modes.
`configGenerator`	`src/lib/services/configGenerator.ts`	Generates `JobConfig` JSON (server IP, tools, domain, licenseKey, registry creds) for provisioning containers. Decrypts stored passwords.
`dockerSpawner`	`src/lib/services/dockerSpawner.ts`	Spawns `docker run` for `letsbe/ansible-runner` containers. Mounts config.json + logs dir. Limits concurrency via `DOCKER_MAX_CONCURRENT`.
`jobService`	`src/lib/services/jobService.ts`	Manages `ProvisioningJob` lifecycle: create, claim (SELECT FOR UPDATE SKIP LOCKED), complete, fail (retry backoff: 1min/5min/15min).
`dnsService`	`src/lib/services/dnsService.ts`	DNS A record verification for all required subdomains per tool before provisioning. Wildcard detection.

Module	Path	Description
`credentialService`	`src/lib/services/credentialService.ts`	AES-256-CBC encrypt/decrypt using `CREDENTIAL_ENCRYPTION_KEY` with scrypt key derivation. Legacy key migration support.
`netcupService`	`src/lib/services/netcupService.ts` (~1150 lines)	Full Netcup SCP API integration via OAuth2 Device Flow. Server list/detail, power actions, metrics, snapshots, rescue mode, hostname, reinstall.
`portainerClient`	`src/lib/services/portainerClient.ts` (~707 lines)	Portainer API client (JWT auth, self-signed cert via undici). Container list/inspect/logs/stats/start/stop/restart/remove.
`stripeService`	`src/lib/services/stripeService.ts`	Stripe webhook verification, checkout session creation, price-to-plan mapping (STARTER/PRO).
`emailService`	`src/lib/services/emailService.ts`	Nodemailer SMTP wrapper. Sends welcome/test/notification emails. Lazy-loads transporter.
`settingsService`	`src/lib/services/settingsService.ts` (~663 lines)	DB-backed system settings in `SystemSetting` table. Encrypted with `SETTINGS_ENCRYPTION_KEY`. Categories: docker, dockerhub, gitea, hub, provisioning, netcup, email, notifications, storage.
`permissionService`	`src/lib/services/permissionService.ts`	Role-based permission matrix (OWNER/ADMIN/MANAGER/SUPPORT) with 20 permission types.
`apiKeyService`	`src/lib/services/apiKeyService.ts`	Hub API key generation (`hk_*` prefix) and SHA-256 hash validation.
`containerHealthService`	`src/lib/services/containerHealthService.ts`	Polls Portainer for container state, detects crashes/OOM/restarts, records `ContainerEvent`.
`errorDetectionService`	`src/lib/services/errorDetectionService.ts`	Regex pattern matching against container log lines using `ErrorDetectionRule` rules.
`statsCollectionService`	`src/lib/services/statsCollectionService.ts`	Collects CPU/memory/disk/network from Netcup + container counts from Portainer. Stores as `ServerStatsSnapshot` (90-day retention).
`storageService`	`src/lib/services/storageService.ts`	S3-compatible (MinIO) storage for staff profile photos.
`totpService`	`src/lib/services/totpService.ts`	TOTP/HOTP via otplib. QR code generation, secret encryption, backup codes.
`securityVerificationService`	`src/lib/services/securityVerificationService.ts`	8-digit codes for destructive actions (WIPE/REINSTALL) on enterprise servers, emailed to contact.

Module	Path	Description
`SSHClient`	`src/lib/ssh/client.ts`	SSH2 wrapper: connect, runCommand, uploadContent (SFTP), streamCommand, testConnection.
`ansible/runner`	`src/lib/ansible/runner.ts`	SSH-based inline provisioning: uploads bootstrap script, installs Docker, deploys orchestrator + sysadmin-agent containers. 30-minute timeout.
`hooks/`	`src/hooks/` (14 files)	React Query hooks for all admin features: analytics, automation, customers, dns, enterprise-clients, netcup, orders, portainer, profile, provisioning-logs, servers, settings, staff, stats, two-factor.

3.2.3 Data Models (Prisma schema — `prisma/schema.prisma`)

Core Business Models:

Model	Table	Key Fields
`User`	`users`	`id`, `email` (unique), `passwordHash`, `name`, `company`, `status` (PENDING_VERIFICATION/ACTIVE/SUSPENDED), `twoFactorEnabled`, `twoFactorSecretEnc`, `backupCodesEnc`
`Staff`	`staff`	`id`, `email` (unique), `passwordHash`, `name`, `role` (OWNER/ADMIN/MANAGER/SUPPORT), `status` (ACTIVE/SUSPENDED), `profilePhotoKey`, 2FA fields
`StaffInvitation`	`staff_invitations`	`id`, `email`, `role`, `token` (unique), `expiresAt`, `invitedBy`
`Subscription`	`subscriptions`	`id`, `userId`, `plan` (TRIAL/STARTER/PRO/ENTERPRISE), `tier` (HUB_DASHBOARD/ADVANCED), `tokenLimit`, `tokensUsed`, `trialEndsAt`, `stripeCustomerId`, `stripeSubscriptionId`, `status`

Model	Table	Key Fields
Order	orders	id, userId, status (8 states: PAYMENT_CONFIRMED→FULFILLED/FAILED), tier, domain, tools[], configJson, automationMode (AUTO/MANUAL/PAUSED), customer, companyName, licenseKey, serverIp, serverPasswordEncrypted, sshPort, netcupServerId, portainerUrl, dashboardUrl, portainerUsername, portainerPasswordEnc, failureReason, timestamps
DnsVerification	dns_verifications	id, orderId (unique), wildcardPassed, manualOverride, allPassed, totalSubdomains, passedCount, lastCheckedAt, verifiedAt
DnsRecord	dns_records	id, dnsVerificationId, subdomain, fullDomain, expectedIp, resolvedIp, status (PENDING/VERIFIED/MISMATCH/NOT_FOUND/ERROR/SK
ProvisioningJob	provisioning_jobs	id, orderId, jobType, status (PENDING/CLAIMED/RUNNING/COMPLETED/FAILED/DEAD), priority, claimedAt, claimedBy, containerName, attempt, maxAttempts (3), nextRetryAt, configSnapshot (JSON), runnerTokenHash, result, error
JobLog	job_logs	id, jobId, level, message, step, progress, timestamp
ProvisioningLog	provisioning_logs	id, orderId, level, message, step, timestamp
TokenUsage	token_usage	id, userId, instanceId, operation, tokensInput, tokensOutput, model, createdAt
RunnerToken	runner_tokens	id, tokenHash (unique), name, isActive, lastUsed
ServerConnection	server_connections	id, orderId (unique), registrationToken (unique), hubApiKey, hubApiKeyHash, orchestratorUrl, agentVersion, status (PENDING/REGISTERED/ONLINE/OFFLINE), lastHeartbeat
RemoteCommand	remote_commands	id, serverConnectionId, type (SHELL/RESTART_SERVICE/UPDATE/ECHO), payload, status (PENDING/SENT/EXECUTING/COMPLETED/FAILED), result, errorMessage, timestamps

Model	Table	Key Fields
SystemSetting	system_settings	id, key (unique), value, encrypted, category

Enterprise/Monitoring Models:

Model	Table	Key Fields
EnterpriseClient	enterprise_clients	id, name, companyName, contactEmail, contactPhone, notes, isActive
EnterpriseServer	enterprise_servers	id, clientId, netcupServerId, nickname, purpose, isActive, portainerUrl, portainerUsername, portainerPasswordEnc
ServerStatsSnapshot	server_stats_snapshots	id, serverId, clientId, timestamp, cpuPercent, memoryUsedMb, memoryTotalMb, diskReadMbps, diskWriteMbps, networkInMbps, networkOutMbps, containersRunning, containersStopped
ErrorDetectionRule	error_detection_rules	id, clientId, name, pattern (regex), severity, isActive, description
DetectedError	detected_errors	id, serverId, ruleId, containerId, containerName, logLine, context, timestamp, acknowledgedAt, acknowledgedBy
SecurityVerificationCode	security_verification_codes	id, clientId, code, action (WIPE/REINSTALL), targetServerId, expiresAt, usedAt, attempts
LogScanPosition	log_scan_positions	id, serverId, containerId, lastLineCount, lastLogHash, lastScannedAt
ContainerStateSnapshot	container_state_snapshots	id, serverId, containerId, containerName, state, exitCode, capturedAt
ContainerEvent	container_events	id, serverId, containerId, containerName, eventType (CRASH/OOM_KILLED/RESTART/STOPPED), exitCode, details, acknowledgedAt
NotificationSetting	notification_settings	id, clientId (unique), enabled, criticalErrorsOnly, containerCrashes, recipients[], cooldownMinutes
NotificationCooldown	notification_cooldowns	id, type (unique), lastSentAt

Model	Table	Key Fields
`Pending2FASession`	`pending_2fa_sessions`	`id`, `token` (unique), `userId`, `userType`, `email`, `name`, `role`, `company`, `subscription`, `expiresAt`

3.2.4 API Endpoints

Authentication:

Method	Path	Description
GET/POST	`/api/auth/[...nextauth]`	NextAuth handlers (Credentials provider)
GET	`/api/v1/auth/invite/[token]`	Look up staff invitation by token
POST	`/api/v1/auth/accept-invite`	Accept staff invitation, set password
GET	`/api/v1/auth/2fa/status`	Get 2FA enabled status
POST	`/api/v1/auth/2fa/setup`	Generate TOTP secret + QR code
POST	`/api/v1/auth/2fa/verify`	Verify TOTP code to enable 2FA
POST	`/api/v1/auth/2fa/disable`	Disable 2FA
GET	`/api/v1/auth/2fa/backup-codes`	Get/regenerate backup codes
POST	`/api/v1/setup`	One-time initial owner account creation

Profile:

Method	Path	Description
GET/PATCH	`/api/v1/profile`	Get/update current user profile
POST	`/api/v1/profile/photo`	Upload profile photo to S3/MinIO
POST	`/api/v1/profile/password`	Change password

Admin — Customers:

Method	Path	Description
GET	`/api/v1/admin/customers`	List customers (pagination, search)
POST	`/api/v1/admin/customers`	Create customer
GET	`/api/v1/admin/customers/[id]`	Get customer detail with orders
PATCH	`/api/v1/admin/customers/[id]`	Update customer

Admin — Orders:

Method	Path	Description
GET	`/api/v1/admin/orders`	List orders (filter by status, tier, search)
POST	`/api/v1/admin/orders`	Create order (staff-initiated, MANUAL mode)
GET	`/api/v1/admin/orders/[id]`	Get order detail
PATCH	`/api/v1/admin/orders/[id]`	Update order (credentials, status)
POST	`/api/v1/admin/orders/[id]/provision`	Spawn Docker provisioning container
GET	`/api/v1/admin/orders/[id]/stream`	SSE stream provisioning logs
GET	`/api/v1/admin/orders/[id]/dns`	Get DNS verification status
POST	`/api/v1/admin/orders/[id]/dns/verify`	Trigger DNS verification
POST	`/api/v1/admin/orders/[id]/dns/override`	Manual DNS override
GET/PATCH	`/api/v1/admin/orders/[id]/automation`	Get/change automation mode
GET	`/api/v1/admin/orders/[id]/containers`	List containers via Portainer
GET	`/api/v1/admin/orders/[id]/container-stats`	All container stats
GET	`/api/v1/admin/orders/[id]/containers/[cId]`	Container detail
POST	`/api/v1/admin/orders/[id]/containers/[cId]/[action]`	start/stop/restart container
GET	`/api/v1/admin/orders/[id]/containers/[cId]/logs`	Container logs
GET	`/api/v1/admin/orders/[id]/containers/[cId]/stats`	Container CPU/memory stats
GET	`/api/v1/admin/orders/[id]/portainer`	Get Portainer credentials
POST	`/api/v1/admin/orders/[id]/portainer`	Initialize Portainer endpoint
POST	`/api/v1/admin/orders/[id]/test-ssh`	Test SSH connection

Admin — Servers:

Method	Path	Description
GET	`/api/v1/admin/servers`	List servers (from fulfilled orders)
GET	`/api/v1/admin/servers/[id]/health`	Server health / connection status
POST	`/api/v1/admin/servers/[id]/cmd`	Queue remote command for orchestrator
POST	`/api/v1/admin/portainer/ping`	Test Portainer connectivity

Admin — Netcup:

Method	Path	Description
GET/POST/DELETE	`/api/v1/admin/netcup/auth`	OAuth2 Device Flow (initiate/poll/disconnect)
GET	`/api/v1/admin/netcup/servers`	List Netcup servers
GET	`/api/v1/admin/netcup/servers/[id]`	Server detail
PATCH	`/api/v1/admin/netcup/servers/[id]`	Power action / hostname / nickname

Method	Path	Description
GET	`/api/v1/admin/netcup/servers/[id]/metrics`	CPU, disk, network metrics
GET/POST	`/api/v1/admin/netcup/servers/[id]/snapshots`	List/create snapshots
DELETE/POST	`/api/v1/admin/netcup/servers/[id]/snapshots`	Delete/revert snapshot

Admin — Enterprise Clients:

Method	Path	Description
GET/POST	`/api/v1/admin/enterprise-clients`	List/create enterprise clients
GET	`/api/v1/admin/enterprise-clients/error-summary`	Aggregated error summary
GET/PATCH/DELETE	`/api/v1/admin/enterprise-clients/[id]`	CRUD
GET	`/api/v1/admin/enterprise-clients/[id]/stats`	Client stats history
GET	`/api/v1/admin/enterprise-clients/[id]/errors`	Detected errors
POST	`/api/v1/admin/enterprise-clients/[id]/errors/[eId]/acknowledge`	Acknowledge error
GET/POST	`/api/v1/admin/enterprise-clients/[id]/error-rules`	Error detection rules CRUD
GET/PATCH/DELETE	`/api/v1/admin/enterprise-clients/[id]/error-rules/[rId]`	Manage individual rule
GET	`/api/v1/admin/enterprise-clients/[id]/error-dashboard`	Error dashboard
GET	`/api/v1/admin/enterprise-clients/[id]/container-events`	Container crash/OOM events
GET/PATCH	`/api/v1/admin/enterprise-clients/[id]/notifications`	Notification settings
GET/POST	`/api/v1/admin/enterprise-clients/[id]/servers`	Servers for client
GET/PATCH/DELETE	`.../servers/[sId]`	Server CRUD
GET	`.../servers/[sId]/stats`	Server stats
GET	`.../servers/[sId]/containers`	Container list
GET	`.../servers/[sId]/containers/[cId]`	Container detail
GET	`.../servers/[sId]/containers/[cId]/logs`	Container logs
POST	`.../servers/[sId]/actions`	WIPE/REINSTALL (with security code)
POST	`.../servers/[sId]/verify`	Verify security code
POST	`.../servers/[sId]/test-portainer`	Test Portainer connection

Admin — Staff:

Method	Path	Description
GET	`/api/v1/admin/staff`	List staff members
POST	`/api/v1/admin/staff/invite`	Send invitation email
GET	`/api/v1/admin/staff/invites`	List pending invitations
DELETE	`/api/v1/admin/staff/invites/[id]`	Cancel invitation
PATCH/DELETE	`/api/v1/admin/staff/[id]`	Update/delete staff

Admin — Settings & Analytics:

Method	Path	Description
GET/POST	`/api/v1/admin/settings`	List all / batch update settings
GET/PATCH	`/api/v1/admin/settings/[key]`	Get/set single setting
POST	`/api/v1/admin/settings/email/test`	Send test email
POST	`/api/v1/admin/settings/storage/test`	Test S3/MinIO
GET	`/api/v1/admin/stats`	Dashboard statistics
GET	`/api/v1/admin/analytics`	Analytics data

Orchestrator Phone-Home (called BY orchestrator instances):

Method	Path	Auth	Description
POST	`/api/v1/orchestrator/register`	`registrationToken` in body	Orchestrator registers post-deploy, receives `hubApiKey`
POST	`/api/v1/orchestrator/heartbeat`	`Bearer {hubApiKey}`	Status heartbeat, credential sync, receive queued commands
GET	`/api/v1/orchestrator/commands`	`Bearer {hubApiKey}`	Poll for pending commands
POST	`/api/v1/orchestrator/commands`	`Bearer {hubApiKey}`	Report command execution result

Public API:

Method	Path	Auth	Description
POST	`/api/v1/public/orders`	`X-API-Key`	Create order from external source (website), AUTO mode
GET	`/api/v1/public/orders/[id]`	`X-API-Key`	Get order status

Webhooks:

Method	Path	Auth	Description
POST	`/api/v1/webhooks/stripe`	Stripe signature	Handles `checkout.session.completed` → creates User, Subscription, Order

Cron:

Method	Path	Auth	Description
GET/POST	`/api/cron/collect-stats`	Bearer `$CRON_SECRET`	Stats + log scanning + container health
GET	`/api/cron/cleanup-stats`	Bearer `$CRON_SECRET`	Purge old stats (90-day retention)

3.2.5 Authentication / Authorization

- **Staff login:** NextAuth.js v5 Credentials provider → bcrypt compare → optional TOTP 2FA (via `Pending2FASession` table, 5-min TTL) → JWT (7-day maxAge, 24h refresh)
- **Customer login:** Same flow, `userType: 'customer'`
- **JWT payload:** `{ id, userType, role, email, name, company, subscription }`
- **Route protection:** NextAuth `auth()` middleware — `/admin/*` requires `userType === 'staff'`
- **Permission matrix:** `permissionService` — 20 permissions across OWNER/ADMIN/MANAGER/SU roles
- **Orchestrator API keys:** `hk_*` prefix, SHA-256 hash storage, `Bearer` header
- **Public API:** `X-API-Key: $PUBLIC_API_KEY` env var comparison
- **Cron:** `Authorization: Bearer $CRON_SECRET`

3.2.6 Configuration

Variable	Description
`DATABASE_URL`	PostgreSQL connection string
`NEXTAUTH_URL`, `NEXTAUTH_SECRET`, `AUTH_TRUST_HOST`	NextAuth config
`CREDENTIAL_ENCRYPTION_KEY`	AES-256-CBC for server passwords
`SETTINGS_ENCRYPTION_KEY`	AES-256-CBC for system settings
`ENCRYPTION_KEY`	Legacy fallback
`HUB_URL`	Public URL of this Hub instance
`ADMIN_EMAIL`, `ADMIN_PASSWORD`	Initial owner setup
`PUBLIC_API_KEY`	External API auth
`CRON_SECRET`	Cron job auth
`STRIPE_SECRET_KEY`, `STRIPE_WEBHOOK_SECRET`	Stripe integration
`STRIPE_STARTER_PRICE_ID`, `STRIPE_PRO_PRICE_ID`	Stripe price IDs
`DOCKER_REGISTRY_URL`, `DOCKER_IMAGE_NAME`, `DOCKER_IMAGE_TAG`	Runner container config

Variable	Description
`DOCKER_NETWORK_MODE`, `DOCKER_MAX_CONCURRENT`	Docker spawning limits
`JOBS_DIR`, `JOBS_HOST_DIR`, `LOGS_DIR`, `LOGS_HOST_DIR`	Job file paths
`TELEMETRY_RETENTION_DAYS`	Stats retention (default 90)

3.3 3. Inter-Repo Dependencies

Target Repo	How	Details
letsbe-ansible-runner	Docker image spawn	`docker-spawner.ts` spawns `code.letsbe.solutions/letsbe/ansible-runner` containers for provisioning jobs. Mounts config.json + logs dir.
letsbe-orchestrator	Deployed on tenant servers via `ansible/runner.ts`	Image `code.letsbe.solutions/letsbe/letsbe-orchest` deployed as container `letsbe-orchestrator-api` on port 8100. Also deploys companion postgres:16 as `letsbe-orchestrator-db`.
letsbe-sysadmin-agent	Deployed on tenant servers via `ansible/runner.ts`	Image `code.letsbe.solutions/letsbe/letsbe-sysadmi` deployed as `letsbe-sysadmin-agent` container. Connected to local orchestrator at `http://letsbe-orchestrator-api:8100`.
letsbe-mcp-browser	**No references found**	Not deployed or referenced anywhere in Hub codebase.

APIs exposed that other repos consume: - `POST /api/v1/orchestrator/register` — called by deployed orchestrators - `POST /api/v1/orchestrator/heartbeat` — called by deployed orchestrators (and by sysadmin agent's `hub_client.py`) - `GET/POST /api/v1/orchestrator/co` — command queue for remote orchestrators - `POST /api/v1/jobs/{jobId}/logs` — called by ansible-runner to stream logs - `PATCH /api/v1/jobs/{jobId}` — called by ansible-runner to update job status - `POST /api/v1/instances/activate` — called by ansible-runner's `local_bootstrap.sh` for license validation

3.4 4. Current State Assessment

Fully implemented and working: - Complete admin dashboard with staff management, RBAC, 2FA - Customer management (CRUD, subscriptions) - Order lifecycle with

8-state automation state machine - Netcup server management (full SCP API integration via OAuth2 Device Flow) - Portainer integration for container management on provisioned servers - DNS verification workflow before provisioning - Docker-based provisioning job spawning with retry logic - SSE log streaming during provisioning - Stripe webhook integration for checkout → order creation - Enterprise client management with per-server monitoring - Error detection with regex-based rules - Container health monitoring (crash/OOM/restart detection) - Email notifications with cooldown - Server stats collection and analytics with 90-day retention - AES-256-CBC credential encryption at rest - Security verification codes for destructive operations - Staff invitation flow with email - S3/MinIO storage for profile photos - System settings management with encryption - SSH-based inline provisioning as alternative path

Partially implemented: - `automationMode: AUTO` — the state machine exists but transition triggers between some states appear to rely on manual admin actions via the dashboard - Token usage tracking (`TokenUsage` model) — schema exists but no clear ingestion path from orchestrators

Not started / planned: - Customer-facing self-service portal (current UI is staff-only admin) - No `README.md` in the repository

Tests: - 10 unit test files under `src/__tests__/unit/lib/` (services: api-key, automation-worker, config-generator, credential, dns, job, permission, security-verification; plus api/client and auth-helpers) - No integration tests, no E2E tests - No coverage configuration or thresholds

3.5 5. External Integrations

Integration	Library	Usage
Netcup SCP API	raw `fetch`	Server management, OAuth2 Device Flow auth
Stripe	`stripe` npm	Checkout sessions, webhooks, plan mapping
Portainer API	`undici`	Container CRUD on provisioned servers (self-signed cert support)
S3 / MinIO	`@aws-sdk/client-s3`	Object storage for profile photos
SMTP	`nodemailer`	Email dispatch
Docker Engine	`child_process.exec('docker run ...')`	Runner container spawning
SSH	`ssh2`	Direct server access for provisioning

4. Repository 2: letsbe-orchestrator

4.1 1. Overview

- **Language/Framework:** Python 3.11, FastAPI >=0.109.0, Uvicorn
- **Approximate LOC:** ~7,500 across ~50 Python source files
- **Key Dependencies:**
  - **ORM:** SQLAlchemy 2.0 async (asyncpg for PostgreSQL, aiosqlite for tests)
  - **Migrations:** Alembic 1.13.0
  - **Validation:** Pydantic v2, pydantic-settings
  - **HTTP:** httpx 0.26.0
  - **Rate Limiting:** slowapi 0.1.9
  - **Testing:** pytest 8.0.0, pytest-asyncio
- **Deployment:**
  - `Dockerfile`: `python:3.11-slim`, installs gcc + libpq-dev, exposes port 8000
  - `docker-compose.yml`: `db` (postgres:16-alpine, port 5434) + `api` (image: `code.letsbe.solutions` port `127.0.0.1:8100:8000`)
  - `docker-compose-dev.yml`: Local build, port 5433, DEBUG=true, ADMIN_API_KEY=`dev-admin-k`
  - `docker-compose.local.yml`: LOCAL_MODE overlay (adds `LOCAL_MODE=true`, `INSTANCE_ID`, `LOCAL_AGENT_KEY`, Hub telemetry vars)
  - `.gitea/workflows/build.yml`: test job (pytest) → build job (Docker push to `code.letsbe.solutions/letsbe/orchestrator`)
  - `deploy.sh`: Registry login, pull, compose up, Alembic migration after 5s delay
  - `nginx.conf`: Reverse proxy to `127.0.0.1:8100`

4.2 2. Architecture

4.2.1 Entry Points

- **Server:** `uvicorn app.main:app --host 0.0.0.0 --port 8000`
- **Migrations:** `alembic upgrade head` (run inside container via `deploy.sh`)
- **Tests:** `pytest -v --tb=short`

4.2.2 Core Modules

Module	Path	Description
`app/main.py`	App factory	Registers all routers under `/api/v1`. CORS middleware, `RequestIDMiddleware` (X-Request-ID), trailing slash normalization, slowapi rate limiting, IntegrityError→409 handler. Lifespan: `LocalBootstrapService.run()` + `HubTelemetryService.start()` on startup.
`app/config.py`	Settings	Pydantic BaseSettings with `@lru_cache` singleton. DATABASE_URL, ADMIN_API_KEY, LOCAL_MODE, HUB_URL, etc.
`app/db.py`	Database	SQLAlchemy async engine factory, session maker, `get_db()` dependency with rollback-on-error.
`app/models/`	ORM models	6 models: Tenant, Agent, Task, Server, Event, RegistrationToken. All use UUID PK + TimestampMixin.
`app/routes/agents.py` (~530 lines)	Agent management	Registration (new token flow + legacy), local registration, heartbeat, list/get agents. Rate limited 5/min on registration.
`app/routes/tasks.py` (~284 lines)	Task CRUD	Create task, list (filter by tenant/status), atomic claim (oldest pending), update status/result.
`app/routes/tenants.py`	Tenant CRUD	Create, list, get, dashboard token set/revoke.
`app/routes/events.py`	Event logging	List, get, create events.
`app/routes/env.py`	Env management	Create ENV_INSPECT/ENV_UPDATE tasks for agent.
`app/routes/files.py`	File management	Create FILE_INSPECT tasks for agent.
`app/routes/health.py`	Health check	Returns `{"status":"ok","version":"..."}`
`app/routes/meta.py`	Instance metadata	Returns `instance_id`, `local_mode`, `version`, `tenant_id`, `bootstrap_status`.
`app/routes/playbooks.py` (~1531 lines)	Playbook orchestration	10 tool playbooks generating COMPOSITE/PLAYWRIGHT tasks. Tools: Chatwoot, Nextcloud, Poste, Keycloak, n8n, Cal.com, Umami, Uptime Kuma, Vaultwarden, Portainer.
`app/routes/registration_tokens.py`	Token management	Admin-only CRUD for registration tokens with SHA-256 hash storage.
`app/playbooks/` (10 files)	Step builders	Each file defines `build_*_steps()` → list of `CompositeStep`, and `create_*_task()` → persists COMPOSITE/PLAYWRIGHT task.

Module	Path	Description
app/services/hub_telemetry.py (~271 lines)	Hub telemetry	Background asyncio service sending periodic JSON to {HUB_URL}/api/v1/instances/{INSTANCE_ID}/telemetry. Windowed SQL aggregates, exponential backoff, ±15% jitter.
app/services/local_bootstrap.py (~168 lines)	Local mode bootstrap	When LOCAL_MODE=true: waits for DB (30 retries × 2s), creates/retrieves "local" tenant. Exposes get_local_tenant_id() and get_bootstrap_status().
app/dependencies/auth.py	Agent auth	get_current_agent() — validates X-Agent-Id + X-Agent-Secret via SHA-256 hash comparison.
app/dependencies/admin_auth.py	Admin auth	verify_admin_api_key() — validates X-Admin-Api-Key header.
app/dependencies/dashboard_auth.py	Dashboard auth	verify_dashboard_token() — validates X-Dashboard-Token per-tenant (defined but not wired to any route).
app/dependencies/local_agent_auth.py	Local agent auth	verify_local_agent_key() — validates X-Local-Agent-Key, returns 404 if not LOCAL_MODE.

4.2.3 Data Models (SQLAlchemy — app/models/)

Table	Key Columns
tenants	id (UUID), name (unique), domain (unique, nullable), dashboard_token_hash (SHA-256, nullable), timestamps
agents	id (UUID), tenant_id (FK→tenants, nullable), name (hostname), version, status (online/offline/invalid), last_heartbeat, token (legacy), secret_hash (SHA-256), registration_token_id (FK), timestamps
tasks	id (UUID), tenant_id (FK→tenants), agent_id (FK→agents, nullable), type (VARCHAR(100), indexed), payload (JSONB), status (pending/running/completed/failed, indexed), result (JSONB, nullable), timestamps
servers	id (UUID), tenant_id (FK→tenants), hostname, ip_address (IPv4/v6), status (provisioning/ready/error/terminated), timestamps
events	id (UUID), tenant_id (FK→tenants, indexed), task_id (FK→tasks, nullable, indexed), event_type (indexed), payload (JSONB), created_at (indexed, immutable)

Table	Key Columns
`registration_tokens`	`id` (UUID), `tenant_id` (FK→tenants, indexed), `token_hash` (SHA-256, indexed), `description`, `max_uses` (default 1, 0=unlimited), `use_count`, `expires_at`, `revoked`, `created_by`, timestamps

Alembic Migrations: 4 migration files — `initial_schema`, `add_agent_fields` (secret_hash, registration_token_id), `add_dashboard_token_hash`, `add_registration_tokens`.

4.2.4 API Endpoints

Method	Path	Auth	Description
GET	`/`	None	Welcome message
GET	`/health`	None	Health check
GET	`/api/v1/meta/instance`	None	Instance metadata (LOCAL_MODE, version, bootstrap)
POST	`/api/v1/tenants`	None	Create tenant
GET	`/api/v1/tenants`	None	List tenants
GET	`/api/v1/tenants/{id}`	None	Get tenant
POST	`/api/v1/tenants/{id}/dashboard-token`	AdminAuth	Set dashboard token
DELETE	`/api/v1/tenants/{id}/dashboard-token`	AdminAuth	Revoke dashboard token
POST	`/api/v1/tasks`	None	Create task
GET	`/api/v1/tasks`	None	List tasks (filter by tenant, status)
GET	`/api/v1/tasks/next`	AgentAuth	Atomically claim oldest pending task
GET	`/api/v1/tasks/{id}`	None	Get task
PATCH	`/api/v1/tasks/{id}`	AgentAuth	Update task status/result
GET	`/api/v1/agents`	None	List agents
GET	`/api/v1/agents/{id}`	None	Get agent
POST	`/api/v1/agents/register`	None (rate limited)	Register agent (token or legacy)
POST	`/api/v1/agents/register-local`	LocalAgentKey	Local agent registration (idempotent)
POST	`/api/v1/agents/{id}/heartbeat`	AgentAuth	Heartbeat
POST	`/api/v1/agents/{id}/env/inspect`	None	Create ENV_INSPECT task
POST	`/api/v1/agents/{id}/env/update`	None	Create ENV_UPDATE task
POST	`/api/v1/agents/{id}/files/inspect`	None	Create FILE_INSPECT task
POST	`/api/v1/tenants/{id}/registration-tokens`	AdminAuth	Create registration token

Method	Path	Auth	Description
GET	`/api/v1/tenants/{id}/registration-tokens`	AdminAuth	List tokens
GET	`/api/v1/tenants/{id}/registration-tokens/{tId}`	AdminAuth	Get token
DELETE	`/api/v1/tenants/{id}/registration-tokens/{tId}`	AdminAuth	Revoke token
GET	`/api/v1/events`	None	List events
GET	`/api/v1/events/{id}`	None	Get event
POST	`/api/v1/events`	AdminAuth	Create event
POST	`/api/v1/tenants/{id}/{tool}/setup`	None	ENV_UPDATE + DOCKER_RELOAD task (10 tools)
POST	`/api/v1/tenants/{id}/{tool}/initial-setup`	None	PLAYWRIGHT task (8 tools)

4.2.5 Authentication

- **Agent Auth (new):** `X-Agent-Id` + `X-Agent-Secret` headers → SHA-256 hash comparison on `agent.secret_hash`
- **Agent Auth (legacy):** `Authorization: Bearer <token>` → plaintext comparison on `agent.token`
- **Admin Auth:** `X-Admin-Api-Key` → comparison against `settings.ADMIN_API_KEY`
- **Dashboard Token (defined, unused):** `X-Dashboard-Token` → SHA-256 comparison against `tenant.dashboard_token_hash`
- **Local Agent Key:** `X-Local-Agent-Key` → comparison against `settings.LOCAL_AGENT_KEY` (returns 404 if not LOCAL_MODE)

4.2.6 Configuration

Variable	Default	Description
`DATABASE_URL`	`postgresql+asyncpg://orchestrator:orchestrator@localhost:5434/orchestrator`	DB connection
`ADMIN_API_KEY`	(required)	Admin endpoint auth
`DEBUG`	`false`	SQL echo + Swagger
`DB_POOL_SIZE/DB_MAX_OVERFLOW/DB_POOL_TIMEOUT/DB_POOL_RECYCLE`	`5/10/30/1800`	Connection pool
`LOCAL_MODE`	`false`	Single-tenant mode
`INSTANCE_ID`	None	Required if LOCAL_MODE
`HUB_URL`	None	Hub telemetry target
`HUB_API_KEY`	None	Hub auth
`HUB_TELEMETRY_ENABLED`	`false`	Enable telemetry
`HUB_TELEMETRY_INTERVAL_SECONDS`	60	Telemetry interval
`LOCAL_TENANT_DOMAIN`	`local.letsbe.cloud`	Auto-created tenant domain
`CORS_ALLOWED_ORIGINS`	`""`	CORS
`LOCAL_AGENT_KEY`	None	Required if LOCAL_MODE

4.3 3. Inter-Repo Dependencies

Target Repo	How	Details
letsbe-hub	HTTP telemetry	`HubTelemetryService` sends `POST {HUB_URL}/api/v1/instances/{INSTANCE_ID}/te` with `X-Hub-Api-Key` header
letsbe-sysadmin-agent	Task consumer	Agent polls `GET /api/v1/tasks/next`, reports via `PATCH /api/v1/tasks/{id}`, heartbeats via `POST /api/v1/agents/{id}/heartbeat`

APIs exposed: - Full REST API consumed by sysadmin-agent (registration, heartbeat, task polling, task updates, events) - Playbook endpoints consumed by Hub dashboard (via remote commands that proxy through orchestrator) - `/api/v1/meta/instance` consumed by ansible-runner's `local_bootstrap.sh`

4.4 4. Current State Assessment

Fully implemented: - Multi-tenant support with UUID-based isolation - Agent registration with both secure (SHA-256 secret hash) and legacy (plaintext token) flows - Registration token system with max_uses, expiration, revocation - Task queue with atomic claim (`SELECT FOR UPDATE` semantics) - 10 tool playbooks generating COMPOSITE and PLAYWRIGHT task chains - Hub telemetry service with windowed SQL aggregates and backoff - LOCAL_MODE bootstrap for single-tenant deployments - Rate limiting on agent registration (5/min) - Request ID middleware for tracing

Partially implemented: - `DashboardAuthDep` defined in `app/dependencies/dashboard_auth.py` but not applied to any route - `servers` table exists but no routes to manage servers directly - Playbook routes have no authentication (security concern — see Gap Analysis)

Not started (from ROADMAP.md): - 8+ additional tool playbooks (NocoDB, Directus, Ghost, MinIO, Activepieces, Listmonk, Odoo, Mixpost) - Server introspection APIs (`/servers/{id}/scan`, `/diagnose`, `/health`) - New task types: NGINX_RELOAD, HEALTHCHECK, STACK_HEALTH - LLM integration for natural language commands - Task chaining based on results - Automatic remediation workflows - Dashboard & UI

Tests: 13 test files covering agent auth, env routes, file routes, task auth, events, hub telemetry, local mode, and 3 playbook tests. Uses in-memory SQLite (not PostgreSQL). No coverage thresholds.

4.5 5. External Integrations

Integration	Target	Details
letsbe Hub	`{HUB_URL}/api/v1/instances/{id}/telemetry`	Periodic telemetry with agent/task/server metrics
Entri DNS	Referenced in CLAUDE.md only	Planned but not implemented
HashiCorp Vault	Referenced in CLAUDE.md only	Planned but not implemented

---

5. Repository 3: letsbe-sysadmin-agent

5.1 1. Overview

- **Language/Framework:** Python 3.11, fully async (asyncio + httpx). No web framework — this is a worker/agent process with no HTTP listener.
- **Approximate LOC:** ~7,600 across 30 source files + ~1,500 lines of tests
- **Key Dependencies:**
  - **HTTP:** httpx >=0.27.0
  - **Logging:** structlog >=24.0.0
  - **Validation:** Pydantic >=2.0.0, pydantic-settings >=2.0.0
  - **Browser:** Playwright 1.49.1 (pinned)
  - **Env:** python-dotenv >=1.0.0
  - **Testing:** pytest >=8.0.0, pytest-asyncio >=0.23.0
- **Deployment:**
  - `Dockerfile`: `python:3.11-slim`, installs docker-cli + Chromium + Docker Compose v2.32.1, `playwright install chromium`, creates non-root `agent` user, `CMD ["python", "-m", "app.main"]`
  - `docker-compose.yml` (dev): Mounts Docker socket + `/opt/letsbe/{env,stacks,nginx}`, runs as `root`, `seccomp=chromium-seccomp.json`, 1.5 CPU / 1GB RAM
  - `docker-compose.local.yml`: LOCAL_MODE overlay
  - `docker-compose.prod.yml`: Pre-built image + MCP browser sidecar service
  - `.gitea/workflows/build.yml`: Docker build+push to `code.letsbe.solutions/letsbe/sysadmin`

5.2 2. Architecture

5.2.1 Entry Points

- **Primary:** `python -m app.main` → `app.main.run()` → `asyncio.run(main())`
- No CLI arguments — all config via environment variables

5.2.2 Core Modules

Module	Path	Description
`app/main.py`	Entry point	Initializes settings, configures logging, validates mounted dirs, creates OrchestratorClient + Agent + TaskManager, starts heartbeat + poll loops, awaits SIGTERM/SIGINT.
`app/config.py`	Settings	Pydantic BaseSettings (frozen). All runtime settings: URLs, credentials, timeouts, security paths, Playwright config. `get_settings()` is `@lru_cache`.
`app/agent.py` (~383 lines)	Agent lifecycle	Registration (priority: persisted creds → LOCAL_MODE → registration token → legacy), heartbeat loop with exponential backoff + jitter, graceful shutdown. Triggers Hub heartbeats after orchestrator heartbeats.
`app/task_manager.py` (~262 lines)	Task dispatch	Polls orchestrator for next task every `poll_interval`s, dispatches to executors via `asyncio.create_task()`, concurrency via `Semaphore(max_concurrent_tasks)`, circuit breaker, status updates (RUNNING→COMPLETED/FAILED).
`app/clients/orchestrator_client.py` (~923 lines)	Orchestrator HTTP client	Circuit breaker (5 failures, 30s cooldown), exponential backoff + jitter, credential persistence to `~/.letsbe-agent/credentials.json` (atomic write, mode 0600), pending result buffering to `~/.letsbe-agent/pending_results.json`. Dual auth: `X-Agent-Id`+`X-Agent-Secret` (new) or `Authorization: Bearer` (legacy).
`app/clients/hub_client.py` (~161 lines)	Hub HTTP client	Optional. Sends heartbeats with tool credentials to `{HUB_URL}/api/v1/orchestrator/heartbeat`. Change detection via SHA-256 hash of credentials.env.
`app/executors/__init__.py`	Executor registry	`EXECUTOR_REGISTRY` dict mapping task type strings → executor classes. `get_executor(task_type)` factory.
`app/executors/base.py`	Base class	`BaseExecutor` (ABC) with `task_type` property + `execute()`. `ExecutionResult` dataclass: `success`, `data`, `error`, `duration_ms`.
`app/executors/echo_executor.py`	Echo	Returns `{"echoed": payload}`. For testing connectivity.

Module	Path	Description
`app/executors/shell_executor.py` (~164 lines)	Shell commands	Validates against `ALLOWED_COMMANDS` allowlist (absolute paths + arg regex). Blocks shell metacharacters. Timeout enforcement.
`app/executors/file_executor.py` (~224 lines)	File write/append	Path traversal prevention, size limits, mode selection (write/append).
`app/executors/file_inspect_executor.py` (~154 lines)	File read	Byte-limited reads with truncation, path security.
`app/executors/env_update_executor.py` (~286 lines)	Env file update	Atomic writes (temp→chmod→rename), key merge/removal, KEY format validation (`^[A-Z][A-Z0-9_]*$`).
`app/executors/env_inspect_executor.py` (~162 lines)	Env file read	Read .env files, optional key filtering, path security.
`app/executors/docker_executor.py` (~291 lines)	Docker Compose	Finds compose file (yml/yaml priority), runs `docker compose up -d --pull` or `--no-pull`, path validation, timeout enforcement.
`app/executors/composite_executor.py` (~208 lines)	Task chains	Executes sequence of sub-tasks. Failure stops chain. Collects per-step results.
`app/executors/nextcloud_executor.py` (~359 lines)	Nextcloud domain	Runs `docker exec` → Nextcloud `occ` commands for trusted domain configuration.
`app/executors/playwright_executor.py` (~330 lines)	Browser automation	Domain validation, scenario lookup + execution, artifact collection (screenshots, traces).
`app/playwright_scenarios/` (8 tools)	Browser scenarios	Initial setup automation for: Cal.com, Chatwoot, Keycloak, n8n, Nextcloud, Poste, Umami, Uptime Kuma. Each ~230-290 lines.
`app/utils/validation.py` (~426 lines)	Security validation	Shell command allowlist, path traversal prevention, ENV key format validation, domain allowlist checking, forbidden metacharacter blocking.
`app/utils/credential_reader.py` (~157 lines)	Credential sync	Reads `/opt/letsbe/env/credentials.env`, extracts structured credentials by tool (Portainer, Nextcloud, Keycloak, MinIO, Poste) for Hub heartbeat. SHA-256 change detection.
`app/utils/logger.py`	Logging	structlog config: JSON (production) or colored console (dev).

5.2.3 Executor Registry (Task Types)

Task Type	Executor Class	Description
ECHO	EchoExecutor	Connectivity test
SHELL	ShellExecutor	Allowlisted shell commands
FILE_WRITE	FileExecutor	Write/append files
FILE_INSPECT	FileInspectExecutor	Read files
ENV_UPDATE	EnvUpdateExecutor	Atomic .env file updates
ENV_INSPECT	EnvInspectExecutor	Read .env files
DOCKER_RELOAD	DockerExecutor	Docker Compose up -d
COMPOSITE	CompositeExecutor	Sequential sub-task chains
NEXTCLOUD_SET_DOMAIN	NextcloudSetDomainExecutor	Nextcloud trusted domain
PLAYWRIGHT	PlaywrightExecutor	Browser automation scenarios

5.2.4 Playwright Scenarios

Scenario	Module	What It Automates
echo	app/playwright_scenarios/echo.py	Test scenario — navigates + screenshots
calcom_initial_setup	app/playwright_scenarios/calcom_initial_setup.py	Admin account signup + onboarding
chatwoot_initial_setup	app/playwright_scenarios/chatwoot_initial_setup.py	Super admin account creation
keycloak_initial_setup	app/playwright_scenarios/keycloak_initial_setup.py	Admin login + realm creation
n8n_initial_setup	app/playwright_scenarios/n8n_initial_setup.py	Owner account setup
nextcloud_initial_setup	app/playwright_scenarios/nextcloud_initial_setup.py	Admin account creation
poste_initial_setup	app/playwright_scenarios/poste_initial_setup.py	Hostname + admin configuration
umami_initial_setup	app/playwright_scenarios/umami_initial_setup.py	Default password change + website tracking
uptime_kuma_initial_setup	app/playwright_scenarios/uptime_kuma_initial_setup.py	Admin account creation

5.2.5 Data Models

Model	Fields	Notes
Task (dataclass)	id, type, payload, tenant_id, created_at	Received from orchestrator
ExecutionResult (dataclass)	success, data, error, duration_ms	Returned by executors
HeartbeatResult (dataclass)	status (enum: SUCCESS/AUTH_FAILED/SERVER_ERROR/NETWORK_ERROR/NOT_REGISTERED), message	

Model	Fields	Notes
`ScenarioOptions` (dataclass)	`timeout_ms`, `screenshot_on_failure/success`, `save_trace`, `allowed_domains`, `artifacts_dir`	
`ScenarioResult` (dataclass)	`success`, `data`, `screenshots`, `error`, `trace_path`	
`Settings` (BaseSettings)	35+ fields (see Configuration below)	Frozen after init

5.2.6 Authentication (Outbound)

- **Orchestrator (new):** `X-Agent-Id` + `X-Agent-Secret` headers. Also `X-Agent-Version` + `X-Agent-Hostname`.
- **Orchestrator (legacy):** `Authorization: Bearer <token>`
- **Hub:** `Authorization: Bearer {HUB_API_KEY}`
- **Credential persistence:** `~/.letsbe-agent/credentials.json` (mode 0600, atomic write)

5.2.7 Configuration

Variable	Default	Description
`ORCHESTRATOR_URL`	`http://host.docker.internal:8000`	Orchestrator base URL
`LOCAL_MODE`	`false`	Single-tenant mode
`LOCAL_AGENT_KEY`	None	Key for local registration
`REGISTRATION_TOKEN`	None	Multi-tenant registration
`AGENT_ID`, `AGENT_SECRET`, `TENANT_ID`	None	Set post-registration
`HUB_URL`, `HUB_API_KEY`	None	Hub credential sync
`HUB_TELEMETRY_ENABLED`	`true`	Enable Hub heartbeats
`HEARTBEAT_INTERVAL`	30s	Heartbeat frequency
`POLL_INTERVAL`	5s	Task poll frequency
`MAX_CONCURRENT_TASKS`	3	Semaphore limit
`CIRCUIT_BREAKER_THRESHOLD/COOLDOWN`	5 failures / 30s	Circuit breaker
`ALLOWED_FILE_ROOT`	`/opt/letsbe`	File ops root
`ALLOWED_ENV_ROOT`	`/opt/letsbe/env`	Env file root
`ALLOWED_STACKS_ROOT`	`/opt/letsbe/stacks`	Docker compose root
`MAX_FILE_SIZE`	10MB	File write limit
`SHELL_TIMEOUT`	60s	Command timeout
`PLAYWRIGHT_DEFAULT_TIMEOUT_MS`	60000	Action timeout
`PLAYWRIGHT_NAVIGATION_TIMEOUT_MS`	120000	Navigation timeout

Variable	Default	Description
`MCP_SERVICE_URL`	None	MCP browser sidecar URL (declare

5.3 3. Inter-Repo Dependencies

Target Repo	How	Details
letsbe-orchestrator	Primary dependency — all API calls	Registration, heartbeat, task polling, result submission, event dispatch. All via `/api/v1/*` paths.
letsbe-hub	Optional — credential sync	Sends heartbeats with Portainer/Nextcloud/Keycloak/MinIO/Poste credentials to `POST /api/v1/orchestrator/heartbeat`.
letsbe-mcp-browser	Sidecar in prod compose	`docker-compose.prod.yml` runs `mcp-browser` at `http://mcp-browser:8931`. `MCP_SERVICE_URL` setting declared but not wired to any executor code yet.

5.4 4. Current State Assessment

Fully implemented: - Agent registration (3 flows: persisted credentials, LOCAL_MODE, registration token) - Heartbeat with exponential backoff + jitter - Task polling with circuit breaker - 10 executor types (ECHO, SHELL, FILE_WRITE, FILE_INSPECT, ENV_UPDATE, ENV_INSPECT, DOCKER_RELOAD, COMPOSITE, NEXTCLOUD_SET_DOMAIN, PLAYWRIGHT) - 8 Playwright initial-setup scenarios - Security: shell command allowlist, path traversal prevention, env key validation, metacharacter blocking - Credential persistence with atomic writes - Pending result buffering for offline recovery - Hub credential sync with SHA-256 change detection

Partially implemented: - `MCP_SERVICE_URL` / `mcp_service_url` setting exists in config but no executor code uses it — the MCP browser integration is declared but not wired

Not started (from ROADMAP.md): - Phase 2: SERVICE_DISCOVER, CONFIG_SCAN, NGINX_INSPECT executors - Phase 3: NGINX_RELOAD, HEALTHCHECK, STACK_HEALTH, PACKAGE_UPGRADE executors - Phase 4: BACKUP, RESTORE, LOG_TAIL, CERT_CHECK executors - MCP sidecar integration for exploratory browser control

Tests: 8 test files covering all executors (composite, docker, env_inspect, env_update, file, file_inspect, nextcloud, playwright). ROADMAP claims 140+ tests. Uses mocks for subprocess/playwright — no real Docker or browser. No integration tests.

5.5 5. External Integrations

Integration	Details
Orchestrator API	All `/api/v1/*` endpoints
Hub API	`POST /api/v1/orchestrator/heartbeat` (optional)
Docker Engine	Docker socket mounted for docker-compose operations
Playwright/Chromium	Local headless browser for initial setup scenarios
MCP Browser	Sidecar at `http://mcp-browser:8931` (prod compose, not yet integrated in code)
Target tools (via Playwright)	Nextcloud, Keycloak, Chatwoot, n8n, Poste, Cal.com, Umami, Uptime Kuma

---

6. Repository 4: letsbe-ansible-runner

6.1 1. Overview

- **Language/Framework:** Pure Bash shell scripts (`set -euo pipefail`). No application framework.
- **Container Base:** `debian:bookworm-slim`
- **Approximate LOC:** ~4,477 (scripts ~3,077, stacks ~800, nginx configs ~600)
- **Key Dependencies (apt):** openssh-client, sshpass, jq, curl, ca-certificates, openssl, zip, unzip
- **Target server installs:** Docker CE, nginx, certbot, fail2ban, ufw, unattended-upgrades, and ~20 more system packages
- **Deployment:**
  - `Dockerfile`: debian:bookworm-slim, copies scripts/stacks/nginx, ENTRYPOINT `entrypoint.sh`
  - `docker-compose.yml`: Used by Hub to spawn short-lived provisioning containers. Image `code.letsbe.solutions/letsbe/ansible-runner`, mounts config.json + logs dir, limits 1.0 CPU / 512M, `restart: "no"`
  - `.gitea/workflows/build.yml`: Docker build+push to `code.letsbe.solutions/letsbe/ansible-r`

6.2 2. Architecture

6.2.1 Entry Points

- **Primary:** `entrypoint.sh` (Docker ENTRYPOINT) — the container's entire lifecycle
- **Remote scripts (executed on target server via SSH/SCP):**
  - `scripts/setup.sh` — Full 10-step server provisioning
  - `scripts/env_setup.sh` — Template rendering + secret generation
  - `scripts/local_bootstrap.sh` — License validation + orchestrator initialization
  - `scripts/backups.sh` — Daily backup cron job
  - `scripts/restore.sh` — Manual backup restoration

6.2.2 Core Modules

Module	Path	Description
`entrypoint.sh` (~323 lines)	Container entry	SSH connects to target server (sshpass + root password), uploads scripts/stacks/nginx via SCP, executes env_setup.sh + setup.sh remotely, streams structured logs to Hub API, PATCHes job status on completion/failure.
`scripts/setup.sh` (~832 lines)	Server provisioning	10-step setup: system packages, Docker CE, disable conflicting services, nginx + fallback config, UFW firewall, optional admin user + SSH key, SSH hardening (port 22022, key-only auth), unattended security updates, deploy tool stacks via docker-compose, deploy sysadmin agent, local_bootstrap.sh, Certbot SSL certs.
`scripts/env_setup.sh` (~678 lines)	Secret generation + template rendering	Accepts customer/domain/company via CLI args or JSON. Generates 50+ cryptographic secrets (passwords, tokens, keys). Replaces `{{ variable }}` placeholders in all docker-compose files, nginx configs, and backup script. Writes master `credentials.env` + `portainer_admin_password.txt`.
`scripts/local_bootstrap.sh` (~259 lines)	Post-deploy bootstrap	Calls `POST {HUB_URL}/api/v1/instances/activate` with license key. Waits for orchestrator health. Gets tenant ID from orchestrator `/api/v1/meta/instance`. Writes `sysadmin-credentials.env` + `admin-credentials.env`.

Module	Path	Description
`scripts/backups.sh` (~473 lines)	Automated backups	Daily 2am cron. Backs up: 18 PostgreSQL databases (pg_dump), 2 MySQL (WordPress, Ghost), 1 MongoDB (LibreChat), env files, nginx configs, rclone config, crontab. Uploads to rclone remote. Rotates: 7 daily local + 4 weekly remote. Writes `backup-status.json`.
`scripts/restore.sh` (~512 lines)	Backup restoration	Subcommands: list, list-remote, download, postgres/mysql/mongo restore per-tool, env restore, nginx restore, full restore.
`nginx/` (33 files)	Nginx configs	Reverse proxy templates for all tools. `{{ variable }}` placeholders. Certbot ACME challenge support.
`stacks/` (28+ dirs)	Docker Compose stacks	Pre-configured stacks for: Activepieces, Cal.com, Chatwoot, Diun+Watchtower, Documenso, Ghost, Gitea, Gitea+Drone, GlitchTip, HTML, Keycloak, LibreChat, Listmonk, MinIO, n8n, Nextcloud, NocoDB, Odoo, Orchestrator, Penpot, Portainer, Redash, Squidex, Sysadmin (agent+MCP browser), Typebot, Umami, Uptime Kuma, Vaultwarden, Windmill, WordPress.

6.2.3 Data Schemas (JSON)

Job Config (`/job/config.json` — mounted by Hub):

```
server: { ip, port, rootPassword }
customer, domain, companyName, licenseKey, dashboardTier
tools: string[]
dockerHub: { username, token, registry? }
gitea: { registry, username, token }
```

Job Result (written to `/tmp/job_result.json`, sent to Hub):

```
dashboard_url, portainer_url, portainer_username?, portainer_password?
```

6.2.4 APIs Called (Outbound)

Method	Path	Auth	Called From
POST	{HUB_API_URL}/api/v1/runner/{JOB_ID}/log	X-Runner-Token	entrypoint.sh:log_to_hub()

Method	Path	Auth	Called From
PATCH	{HUB_API_URL}/api/v1/runner/{JOB_ID}	X-Runner-Token	entrypoint.sh:update_job_status()
POST	{HUB_URL}/api/v1/instances/activate	license_key in body	local_bootstrap.sh:validate_license()
GET	{ORCHESTRATOR_URL}/health	None	local_bootstrap.sh:wait_for_orchestrator()
GET	{ORCHESTRATOR_URL}/api/v1/meta/instance	None	local_bootstrap.sh:get_local_tenant_id()

6.2.5 Authentication

- **Runner → Hub:** `X-Runner-Token: ${RUNNER_TOKEN}` header on all job log/status endpoints
- **Runner → Target Server:** `sshpass` with root password from config.json. SSH options: `StrictHostKeyChecking=no`
- **Bootstrap → Hub:** License key in JSON body
- **Post-setup SSH:** Hardened to port 22022, key-only auth (`PermitRootLogin prohibit-password`)

6.2.6 Configuration

Variable	Default	Description
`HUB_API_URL`	`https://hub.letsbe.solutions`	Hub API base URL
`JOB_ID`	(empty)	Job identifier
`RUNNER_TOKEN`	(empty)	Hub auth token
`JOB_CONFIG_PATH`	`/job/config.json`	Config file path

6.3 3. Inter-Repo Dependencies

Target Repo	How	Details
letsbe-hub	HTTP API calls	Streams logs, updates job status, activates license.
letsbe-orchestrator	Deploys + bootstraps	`stacks/orchestrator/docker-compose.yml` pulls `code.letsbe.solutions/letsbe/orchestrator:l` `local_bootstrap.sh` calls orchestrator's `/health` and `/api/v1/meta/instance`.
letsbe-sysadmin-agent	Deploys as stack	`stacks/sysadmin/docker-compose.yml` pulls `code.letsbe.solutions/letsbe/sysadmin-agent` Always deployed in step 9.5 of `setup.sh`.

Target Repo	How	Details
letsbe-mcp-browser	Deploys as sidecar	`stacks/sysadmin/docker-compose.yml` also runs `code.letsbe.solutions/letsbe/mcp-browser:la` as `mcp-browser` service on port 8931.

6.4 4. Current State Assessment

Fully implemented: - Complete server provisioning pipeline (10 steps) - Template rendering for 33 nginx configs and 28+ docker-compose stacks - 50+ cryptographic secret generation - SSH hardening (port change, key-only auth, fail2ban) - UFW firewall configuration - Docker CE installation - Certbot SSL certificate issuance - Automated backup system (PostgreSQL × 18, MySQL × 2, MongoDB × 1) - Backup restoration for all supported databases - Backup rotation (7 daily + 4 weekly) - Hub integration (log streaming, status updates, license activation) - Orchestrator bootstrap (health check, tenant ID retrieval)

Bugs/Gaps identified: - `stacks/sysadmin/docker-compose.yml` line 66: `MCP_BROWSER_API_KEY={{ mcp_browser_api_key }}` template variable is referenced but NOT generated in `env_setup.sh`'s sed substitutions — this placeholder would survive to runtime unchanged

Not started: - No tests whatsoever - No README

6.5 5. External Integrations

Integration	Details
ifconfig.co / icanhazip.com	Public IP detection during env_setup
Let's Encrypt / Certbot	SSL certificate issuance
Docker Hub	Public images (WordPress, Ghost, Redis, Postgres, etc.)
ghcr.io	LibreChat, Windmill, Nextcloud images
quay.io	Keycloak images
code.letsbe.solutions	Private Gitea registry for LetsBe images
rclone remote	Cloud backup storage
LibreChat AI providers	Groq, Mistral, OpenRouter, Portkey, Anthropic, OpenAI, Google Gemini (configured in `librechat.yaml`)

7. Repository 5: letsbe-mcp-browser

7.1 1. Overview

- **Language/Framework:** Python 3.11, FastAPI >=0.109.0, Uvicorn
- **Approximate LOC:** ~1,246 (927 source + 319 tests)
- **Key Dependencies:** FastAPI, Uvicorn, Playwright 1.56.0 (pinned), Pydantic v2, pydantic-settings, pytest + pytest-asyncio + httpx (test)
- **Deployment:**
  - `Dockerfile`: `mcr.microsoft.com/playwright/python:v1.56.0-jammy` base (Chromium pre-installed), exposes port 8931, healthcheck via `/health`
  - `docker-compose.yml`: Dev stack, port `8931:8931`, `seccomp=chromium-seccomp.json`, 1.5 CPU / 1G RAM, 3 max sessions
  - `.gitea/workflows/build.yml`: test job (pytest) → build job (Docker push to `code.letsbe.solutions/letsbe/mcp-browser`)

7.2 2. Architecture

7.2.1 Entry Points

- **Server:** `uvicorn app.server:app --host 0.0.0.0 --port 8931`
- **Lifespan:** On startup → launch Chromium, init SessionManager, start cleanup task. On shutdown → close all sessions, stop browser.

7.2.2 Core Modules

Module	Path	Description
`app/server.py` (~451 lines)	FastAPI app	All HTTP endpoints, Pydantic request/response models, API key auth dependency, lifespan management.
`app/config.py` (~38 lines)	Settings	Pydantic BaseSettings: MAX_SESSIONS (3), IDLE_TIMEOUT (300s), MAX_SESSION_LIFETIME (1800s), MAX_ACTIONS_PER_SESSION (50), API_KEY, SCREENSHOTS_DIR.
`app/session_manager.py` (~260 lines)	Session management	`BrowserSession` (isolated session with domain filter, page, timestamps, action counter). `SessionManager` (dict of sessions, asyncio.Lock, background cleanup loop).
`app/playwright_client.py` (~88 lines)	Browser singleton	Single Chromium instance shared across sessions. Docker-compatible launch flags (–no-sandbox, –disable-gpu, –single-process).

Module	Path	Description
`app/domain_filter.py` (~83 lines)	URL allowlist	Exact domain, wildcard subdomain (`*.example.com`), domain with port. Case-insensitive regex compilation.

7.2.3 API Endpoints

Method	Path	Auth	Description
GET	`/health`	None	Health check: status, active_sessions, max_sessions
POST	`/sessions`	API Key	Create session with allowed_domains list. Returns session_id, timestamps, action limits.
GET	`/sessions/{id}/status`	API Key	Session status
DELETE	`/sessions/{id}`	API Key	Close session (idempotent)
POST	`/sessions/{id}/navigate`	API Key	Navigate to URL (domain allowlist checked). Returns title, url, or blocked reason.
POST	`/sessions/{id}/click`	API Key	Click element by CSS selector
POST	`/sessions/{id}/type`	API Key	Fill text into selector, optional Enter press
POST	`/sessions/{id}/wait`	API Key	Wait for selector/text/timeout
POST	`/sessions/{id}/screenshot`	API Key	Take PNG screenshot, save to disk
POST	`/sessions/{id}/snapshot`	API Key	Accessibility tree as flat node list (ref, role, name, text)
GET	`/metrics`	API Key	Basic metrics: active sessions, max sessions

7.2.4 Authentication

- **Optional API key:** `X-API-Key` header. If `API_KEY` env var is empty, auth is disabled. `/health` is always public.

7.2.5 Configuration

Variable	Default	Description
`MAX_SESSIONS`	3	Max concurrent browser sessions
`IDLE_TIMEOUT_SECONDS`	300 (5 min)	Session idle timeout

Variable	Default	Description
MAX_SESSION_LIFETIME_SECONDS	1800 (30 min)	Absolute session lifetime
MAX_ACTIONS_PER_SESSION	50	Action limit per session
BROWSER_HEADLESS	True	Headless mode
DEFAULT_TIMEOUT_MS	30000	Element timeout
NAVIGATION_TIMEOUT_MS	60000	Navigation timeout
API_KEY	"" (disabled)	API key for auth
SCREENSHOTS_DIR	/screenshots	Screenshot storage

7.3 3. Inter-Repo Dependencies

Target Repo	How	Details
None	—	This is a standalone sidecar. No outbound calls to other repos.

Consumed by: - **letsbe-sysadmin-agent** (docker-compose.prod.yml): Runs as sidecar at http://mcp-browser:8931 with shared MCP_BROWSER_API_KEY - **letsbe-ansible-runner** (stacks/sysadmin/docker-compose.yml): Deployed alongside sysadmin agent

7.4 4. Current State Assessment

Fully implemented: - Session-based browser management with domain allowlisting - All core browser actions (navigate, click, type, wait, screenshot, snapshot) - Automatic session cleanup (idle timeout, max lifetime, action limits) - Security via mandatory domain restrictions - Docker deployment with Chromium seccomp profile

Tests: 10 unit tests for DomainFilter, 14 for SessionManager (sync + async mocks). No integration tests, no FastAPI TestClient tests. httpx test dependency unused.

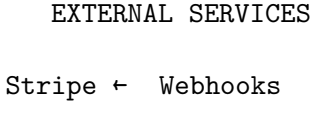
Zero TODOs or stubs in the codebase.

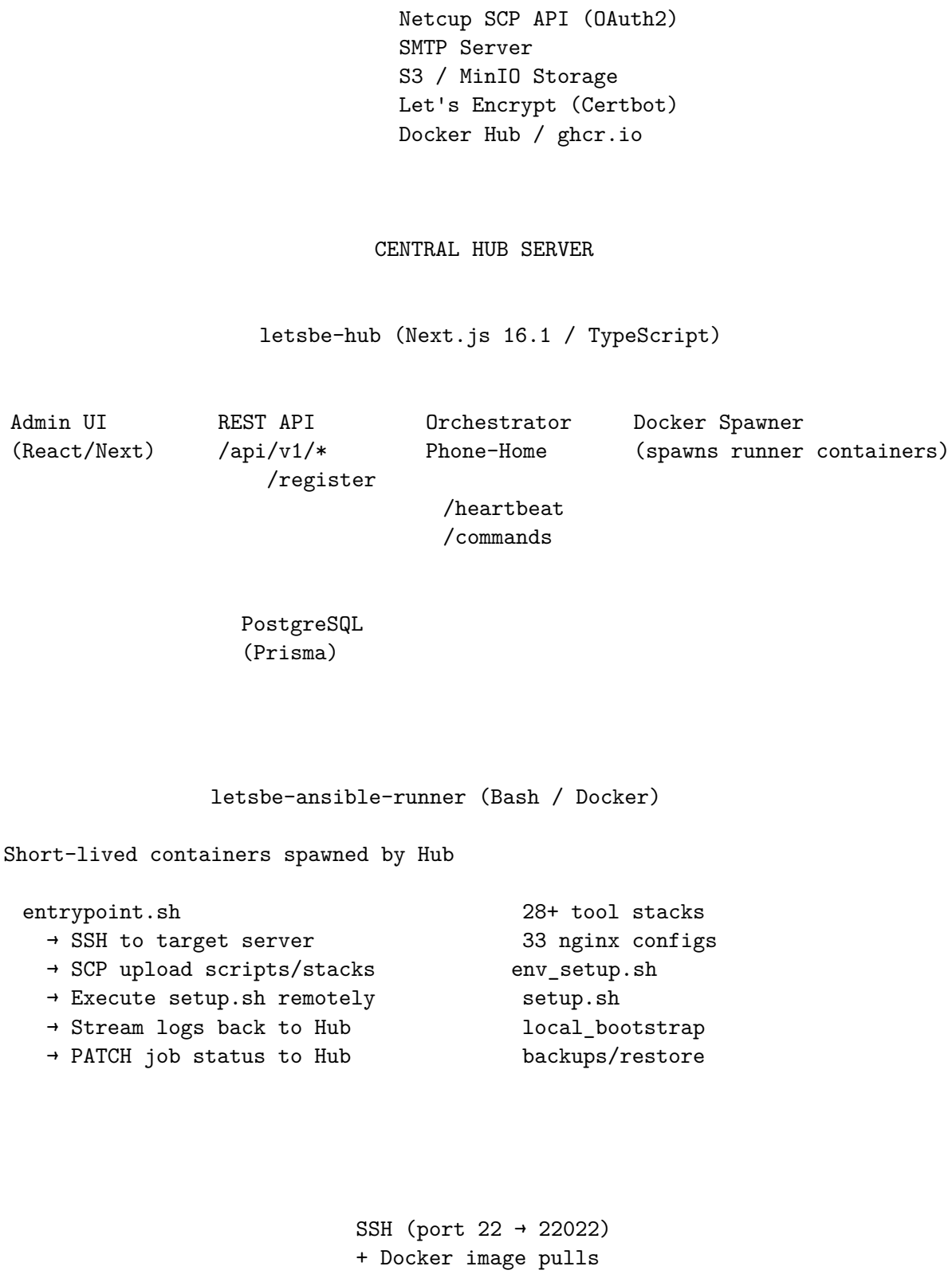
7.5 5. External Integrations

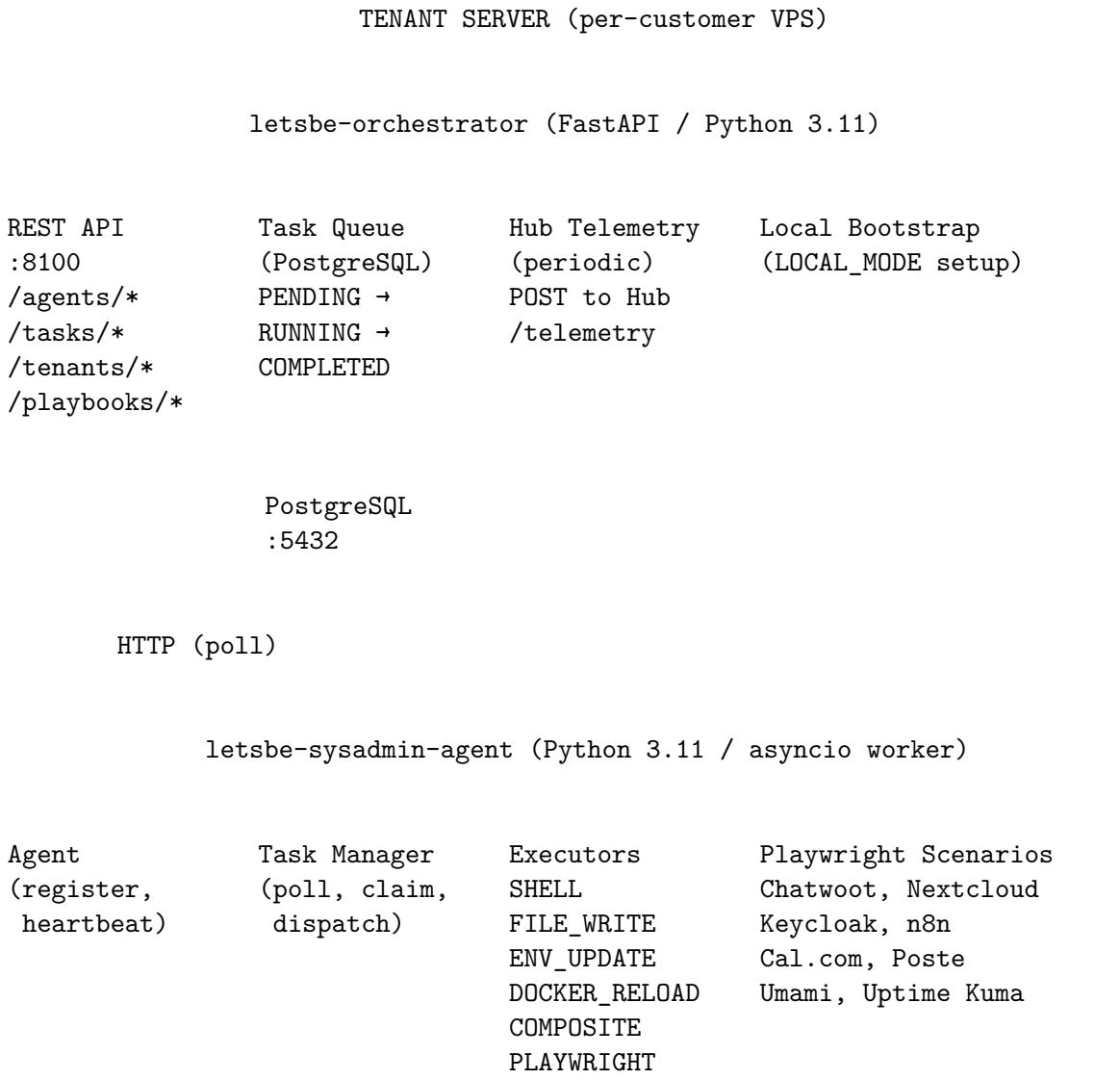
None. This service is entirely self-contained. It controls only a local Chromium browser instance. No external API calls.

---

8. 6. System Architecture Map




Netcup SCP API (OAuth2)
SMTP Server
S3 / MinIO Storage
Let's Encrypt (Certbot)
Docker Hub / ghcr.io
CENTRAL HUB SERVER
letsbe-hub (Next.js 16.1 / TypeScript)
Admin UI
(React/Next)
REST API
/api/v1/*
/register
Orchestrator
Phone-Home
/heartbeat
/commands
Docker Spawner
(spawns runner containers)
PostgreSQL
(Prisma)
letsbe-ansible-runner (Bash / Docker)
Short-lived containers spawned by Hub
entrypoint.sh
→ SSH to target server
→ SCP upload scripts/stacks
→ Execute setup.sh remotely
→ Stream logs back to Hub
→ PATCH job status to Hub
28+ tool stacks
33 nginx configs
env_setup.sh
setup.sh
local_bootstrap
backups/restore
SSH (port 22 → 22022)
+ Docker image pulls


TENANT SERVER (per-customer VPS)
letsbe-orchestrator (FastAPI / Python 3.11)
REST API
:8100
/agents/*
/tasks/*
/tenants/*
/playbooks/*
Task Queue
(PostgreSQL)
PENDING →
RUNNING →
COMPLETED
Hub Telemetry
(periodic)
POST to Hub
/telemetry
Local Bootstrap
(LOCAL_MODE setup)
PostgreSQL
:5432
HTTP (poll)
letsbe-sysadmin-agent (Python 3.11 / asyncio worker)
Agent
(register,
heartbeat)
Task Manager
(poll, claim,
dispatch)
Executors
SHELL
FILE_WRITE
ENV_UPDATE
DOCKER_RELOAD
COMPOSITE
PLAYWRIGHT
Playwright Scenarios
Chatwoot, Nextcloud
Keycloak, n8n
Cal.com, Poste
Umami, Uptime Kuma
letsbe-mcp-browser (FastAPI / Playwright sidecar)
:8931
Session Mgmt
(create,
cleanup)
Domain Filter
(allowlist)
Chromium
(headless)
navigate,
HTTP API
(not yet wired from agent)

```
                                    click, type
                          screenshot
                                      snapshot



                          DEPLOYED TOOL STACKS (Docker Compose)

    Portainer     Nextcloud     Keycloak     n8n     Cal.com     WordPress     ...
    Chatwoot      Poste         MinIO        Ghost   Umami       LibreChat
    Typebot       NocoDB        Vaultwarden  Penpot   Uptime K    Redash


  nginx (reverse proxy for all tools) + Certbot SSL
  UFW firewall + fail2ban + SSH hardening (port 22022)


PROTOCOLS:
  Hub  [HTTP REST]   Ansible Runner (spawned Docker container)
  Ansible Runner  [SSH/SCP]   Tenant Server (provisioning)
  Ansible Runner  [HTTP REST]   Hub (log streaming, job status)
  Orchestrator  [HTTP REST]   Hub (telemetry, registration)
  Sysadmin Agent  [HTTP REST]   Orchestrator (register, heartbeat, tasks)
  Sysadmin Agent  [HTTP REST]   Hub (credential sync heartbeat)
  Sysadmin Agent  [HTTP REST]   MCP Browser (not yet wired)
  Hub Admin UI  [HTTP REST]   Hub API (same process, /api/v1/*)
  Hub  [Portainer API/HTTPS]   Tenant Portainer (container management)
  Hub  [SSH]   Tenant Server (direct SSH, test-connection)
```

---

9. 7. Data Flow

9.1 7.1 User Signs Up and Logs into the Hub

1. **Stripe Checkout:** Customer completes payment on external website. Stripe sends `checkout.session.completed` webhook to `POST /api/v1/webhooks/stripe` (`src/app/api/v1/we`
2. **Webhook Handler** (`stripeService.ts`): Verifies Stripe signature, extracts `customer_email`, `customer_name`, `line_items`, `metadata` (domain, tools). Maps price ID to plan (STARTER→ADVANCED PRO→HUB_DASHBOARD).
3. **Create User + Subscription + Order**: Creates `User` record (bcrypt-hashed password from metadata or generated), `Subscription` (plan, tier, limits), and `Order` (status: PAYMENT_CONFIRMED, domain, tools, tier, automationMode: AUTO).

4. **Staff Login (Admin):** Staff navigates to `/login`. `login-form.tsx` posts credentials to NextAuth `/api/auth/callback/credentials`. NextAuth `authorize()` in `src/lib/auth.ts` looks up `Staff` by email, compares bcrypt hash. If 2FA enabled: creates `Pending2FASession` (5-min TTL), returns `2FA_REQUIRED:{token}`. Frontend shows TOTP input. Second `authorize()` call verifies TOTP via `totpService`, issues JWT (7-day maxAge). Redirect to `/admin`.

9.2 7.2 User Creates and Configures an AI Agent

The platform doesn't have a direct "create agent" user flow. Agents are automatically deployed during server provisioning. The flow is:

1. **Admin creates Order** (or AUTO mode from Stripe): Order enters the `automationWorker` state machine (`automation-worker.ts`).
2. **State: AWAITING_SERVER**: Admin assigns a Netcup server (via dashboard, `PATCH /api/v1/admin/orders/[id]` with `serverIp`). Or selects from Netcup server list.
3. **State: SERVER_READY → DNS_PENDING → DNS_READY**: `dnsService` verifies A records for all tool subdomains. Admin can skip via `POST /dns/skip`.
4. **State: PROVISIONING**: `POST /api/v1/admin/orders/[id]/provision` triggers `dockerSpawner.ts` which runs `docker run letsbe/ansible-runner` with the job config.
5. **Runner provisions server** (see 7.4 below). Among other things, deploys the orchestrator and sysadmin-agent containers.
6. **Agent auto-registers:** On first boot, `letsbe-sysadmin-agent` calls `POST /api/v1/agents/registe` on the local orchestrator, receives `agent_id` + `agent_secret`, persists to `~/.letsbe-agent/credenti`
7. **Agent starts polling:** `task_manager.py` polls `GET /api/v1/tasks/next` every 5 seconds. Orchestrator can now dispatch tasks.

9.3 7.3 Agent Executes a Task (Trigger to Completion)

1. **Task Creation:** Hub admin (or playbook endpoint) calls `POST /api/v1/tasks` on the orchestrator with `{tenant_id, type, payload}`. Example: `type: "COMPOSITE", payload: { sequence: [{task: "ENV_UPDATE", payload: {...}}, {task: "DOCKER_RELOAD", payload: {...}}] }`.
2. **Task Queued:** Orchestrator persists to `tasks` table with `status: PENDING`.
3. **Agent Claims:** `sysadmin-agent`'s `task_manager.py` calls `GET /api/v1/tasks/next`. Orchestrator atomically claims oldest pending task for the agent's tenant (or any tenant if agent has no tenant_id). Returns `Task(id, type, payload)`.
4. **Agent Dispatches:** `task_manager.py` looks up executor in `EXECUTOR_REGISTRY[task.type]`, wraps in `asyncio.create_task()` (bounded by `Semaphore(3)`).
5. **Agent Reports RUNNING:** `PATCH /api/v1/tasks/{id}` with `{status: "running"}`.
6. **Executor Runs:**
   - `COMPOSITE`: Iterates through `payload.sequence`, executing each sub-task's executor in order. Stops on first failure.
   - `ENV_UPDATE`: Reads .env file, merges new key-value pairs, atomic write (temp→chmod→renar

- `DOCKER_RELOAD`: Finds docker-compose.yml in specified dir, runs `docker compose up -d --pull`.
- `PLAYWRIGHT`: Looks up scenario by name, launches Chromium page with domain allowlist, executes scenario steps (click, fill, wait, screenshot), collects artifacts.

7. **Agent Reports Result:** `PATCH /api/v1/tasks/{id}` with `{status: "completed", result: {...}}` or `{status: "failed", error: "..."}`.
8. **Offline Recovery:** If orchestrator is unreachable when reporting, result is buffered to `~/.letsbe-agent/pending_results.json` and retried on next successful connection.

9.4 7.4 Ansible Playbook Gets Triggered and Runs

1. **Trigger:** Hub admin clicks "Provision" → `POST /api/v1/admin/orders/[id]/provision` (in Hub).
2. **Config Generation:** `configGenerator.ts` builds `JobConfig` JSON: server IP, root password (decrypted), domain, tools list, customer name, license key, Docker/Gitea registry credentials.
3. **Job Creation:** `jobService.ts` creates `ProvisioningJob` (status: PENDING). Generates `RunnerToken` for auth.
4. **Docker Spawn:** `dockerSpawner.ts` runs: `docker run --name letsbe-runner-{jobId} -v {configPath}:/job/config.json:ro -v {logsPath}:/logs --memory=512m --cpus=1.0 code.letsbe.solutions/letsbe/ansible-runner:latest`
5. **Container Starts → `entrypoint.sh`:**
   - `load_config()`: Parses `/job/config.json`, extracts server IP, credentials, tools list.
   - `step_prepare()`: SSH to target, creates `/opt/letsbe/{env,stacks,nginx,config,scripts}` directories.
   - `step_docker_login()`: SSH, runs `docker login` for Gitea registry and Docker Hub.
   - `step_upload()`: SCP uploads all `scripts/`, `stacks/`, `nginx/` directories to `/opt/letsbe/`.
   - `step_env()`: SSH, runs `env_setup.sh --customer X --domain Y --company Z`. Generates 50+ secrets, replaces all `{{ variable }}` templates.
   - `read_portainer_credentials()`: SSH, reads `credentials.env` for Portainer admin user/password.
   - `step_setup()`: SSH, runs `setup.sh --tools "tool1,tool2,..." --domain "example.com"`. This is the big 10-step provisioning (packages, Docker, nginx, firewall, SSH hardening, tool stacks, sysadmin agent, bootstrap, SSL).
   - `step_finalize()`: Writes `/tmp/job_result.json`, PATCHes job status to Hub as `completed` (or `failed`).
6. **Log Streaming:** Throughout, `log_to_hub()` sends `POST {HUB_API_URL}/api/v1/jobs/{JOB_ID}/lo` with structured JSON entries. Hub SSE endpoint (`GET /orders/[id]/logs/stream`) streams these to the admin UI.
7. **Container Exits:** `restart: "no"` — container stops when `entrypoint.sh` finishes.

9.5 7.5 MCP Browser Tool Gets Invoked by an Agent

Current state: NOT YET FULLY WIRED.
The intended flow (based on code analysis):

1. **Task Creation:** Orchestrator creates a task (likely future `BROWSER_EXPLORE` type or enhanced `PLAYWRIGHT` type) with a payload specifying URL and actions.
2. **Agent Receives Task:** `PlaywrightExecutor` (or a future MCP executor) gets the task.
3. **Agent Calls MCP Browser:** HTTP calls to `http://mcp-browser:8931`:
   - `POST /sessions` — creates session with `allowed_domains`
   - `POST /sessions/{id}/navigate` — navigate to URL
   - `POST /sessions/{id}/snapshot` — get accessibility tree
   - `POST /sessions/{id}/click`, `/type`, `/wait` — interact with page
   - `POST /sessions/{id}/screenshot` — capture visual state
   - `DELETE /sessions/{id}` — cleanup
4. **Agent Returns Result:** Compiles browser interaction results and sends back to orchestrator.

What's actually implemented today: - MCP browser service is fully functional with all HTTP endpoints - MCP browser is deployed as a sidecar alongside the sysadmin agent (in `stacks/sysadmin/docker-compose.yml` and `docker-compose.prod.yml`) - `MCP_SERVICE_URL` setting exists in sysadmin-agent config but is **not connected to any executor** - The agent currently uses its own embedded Playwright (via `PlaywrightExecutor` + `playwright_scenarios/`) rather than the MCP browser sidecar - The MCP browser appears designed for future LLM-driven exploratory browsing (as opposed to the scripted scenarios the agent runs today)

---

10. 8. Gap Analysis

10.1 8.1 Missing Pieces for Production SaaS Launch

1. **Customer-facing portal:** The Hub UI is staff-only admin. No self-service portal for customers to manage their servers, view status, or configure tools. The `User` model and `Subscription` model exist but are only used in the Stripe webhook flow.
2. **Automated end-to-end provisioning:** The `automationWorker` state machine exists but transitions between states (AWAITING_SERVER → SERVER_READY → DNS_PENDING → DNS_READY) appear to require manual admin actions. True AUTO mode needs: automatic Netcup server allocation, automatic DNS configuration (Entri integration referenced but not built), and automatic progression through all states.

3. **DNS automation:** `dnsService` only verifies DNS records — it doesn't create them. An external DNS provider integration (Entri, Cloudflare, etc.) is needed to automatically configure A records for customer domains.
4. **Monitoring alerting completeness:** Container health monitoring exists for enterprise clients but is not integrated into the standard order/server management flow. Self-hosted customers have no monitoring dashboard.
5. **Customer notification system:** Email welcome + security codes exist, but no customer-facing status notifications (provisioning progress, downtime alerts, cert renewal warnings).
6. **Backup management UI:** Backups are configured via cron on the target server (`backups.sh`) but there's no admin UI to monitor backup status, trigger restores, or configure backup settings.
7. **Multi-server support per customer:** The current model is 1 order = 1 server. No support for customers needing multiple servers or high-availability deployments.
8. **Rate limiting / abuse prevention:** Only agent registration is rate-limited (5/min). Public API, webhook endpoints, and admin routes have no rate limiting.
9. **Audit logging:** Events table exists in orchestrator but no comprehensive audit trail in the Hub for admin actions (who changed what, when).
10. **API documentation:** No OpenAPI/Swagger docs served in production (Swagger only enabled when DEBUG=true in orchestrator).

10.2 8.2 Security Concerns

1. **Hardcoded dev credentials:** `docker-compose-dev.yml` in orchestrator has `ADMIN_API_KEY=dev-a` Risk: if accidentally used in production.
2. **Unauthenticated orchestrator endpoints:** Multiple endpoints lack auth:
   - `POST /api/v1/tenants` — anyone can create tenants
   - `POST /api/v1/tasks` — anyone can create tasks
   - `GET /api/v1/tasks`, `GET /api/v1/agents` — anyone can list tasks/agents
   - All playbook endpoints (`POST /api/v1/tenants/{id}/{tool}/setup`) — no auth
   - This is partially mitigated by the orchestrator only listening on `127.0.0.1:8100`, but any process on the tenant server can access it.
3. **Root password in transit:** The ansible-runner receives root passwords via `config.json` mounted from the Hub server. The password travels: Hub DB (encrypted) → `configGenerator` (decrypted) → config.json file on Hub disk → Docker mount → entrypoint.sh → sshpass over SSH. The config.json file sits unencrypted on disk.

4. **SSH StrictHostKeyChecking=no:** `entrypoint.sh` disables host key checking, making it vulnerable to MITM attacks during initial provisioning.
5. **MCP browser API key gap:** `stacks/sysadmin/docker-compose.yml` references `{{ mcp_browser_api_key }}` but `env_setup.sh` never generates or substitutes this variable. The MCP browser would run with no API key in production.
6. **Docker socket access:** Both the sysadmin agent and Hub production containers mount `/var/run/docker.sock` and run as root, giving full Docker engine access. The sysadmin agent has a TODO about using Docker group membership instead.
7. **Secrets in .env files:** All credentials are stored in plaintext `.env` files on the target server at `/opt/letsbe/env/`. No at-rest encryption. The ROADMAP mentions Vault integration but it's not built.
8. **No CSRF protection:** NextAuth provides some CSRF via JWT, but custom API routes don't have explicit CSRF tokens.
9. **NextAuth beta:** Using `next-auth@5.0.0-beta.30` — a pre-release version in production.

10.3 8.3 Scalability Bottlenecks

1. **Single Hub instance:** No horizontal scaling story for the Hub. PostgreSQL connection pooling exists but no multi-instance deployment config.
2. **Docker spawner concurrency:** `DOCKER_MAX_CONCURRENT=3` limits provisioning parallelism. Spawning Docker containers on the Hub server itself limits vertical scale.
3. **Ansible runner is synchronous SSH:** Each provisioning job holds an SSH connection for 10-30 minutes. No work distribution or queue beyond Docker container isolation.
4. **Orchestrator per-tenant:** Every tenant server runs its own PostgreSQL + orchestrator + sysadmin agent. This is clean isolation but expensive per-customer (3 containers overhead before any tools).
5. **Stats collection cron:** `collect-stats` cron job polls every Netcup server and Portainer instance serially. As enterprise client count grows, this becomes a bottleneck.
6. **Portainer API as single point:** All container management goes through Portainer's HTTP API. If Portainer is down on a server, no container visibility.

10.4 8.4 Overlap / Duplication

1. **Dual provisioning paths:** The Hub has both `ansible/runner.ts` (SSH-based inline provisioning) AND `docker-spawner.ts` (ansible-runner container). The SSH path appears to be an older alternative. Both deploy the same infrastructure.

2. **Playwright in two places:** The sysadmin agent has its own embedded Playwright (with `playwright_scenarios/`), AND the MCP browser sidecar provides an HTTP API for Playwright. The agent doesn't use the MCP browser yet — it runs Chromium directly.
3. **Credential sync duplication:** Credentials flow Hub → ansible-runner → `credentials.env` on server → sysadmin-agent reads and sends back to Hub via heartbeat. This round-trip is redundant since the Hub generated the credentials in the first place.
4. **Hub telemetry vs credential heartbeat:** The orchestrator sends telemetry to Hub (`hub_telemetry.py`), AND the sysadmin agent independently sends heartbeats with credentials to Hub (`hub_client.py`). Two separate phone-home mechanisms to the same Hub.

10.5 8.5 Dead Code / Potentially Unnecessary Components

1. `DashboardAuthDep` in orchestrator: Defined but not applied to any route. Dashboard token can be set/revoked via admin endpoints but never checked.
2. `servers` **table in orchestrator:** Model exists, relationships defined, but no routes to manage servers. Status values defined (provisioning/ready/error/terminated) but never set by any code path.
3. **LibreChat MCP server configs:** `stacks/librechat/librechat.yaml` has commented-out MCP server configurations that are non-functional.
4. **Legacy agent token auth:** Both orchestrator and sysadmin-agent maintain backward-compatible legacy token auth alongside the new secret-hash auth. This adds complexity.
5. `ansible/runner.ts` **in Hub:** The SSH-based inline provisioning path appears to be superseded by the Docker-based ansible-runner approach, but the code remains.

---

11. 9. Tech Debt & Quick Wins

11.1 Top 10 Tech Debt Items

#	Item	Repo	Impact	Effort
1	**Unauthenticated orchestrator endpoints** — tenant/task CRUD and all playbook routes have no auth. Any process on the tenant server can create tasks or trigger playbooks.	orchestrator	HIGH (security)	Medium — add AdminAuth or DashboardAuth dependency to routes
2	**No integration/E2E tests across any repo** — all tests are unit tests with mocks. No tests that verify actual HTTP flows, database interactions (Hub uses mocked Prisma, orchestrator uses SQLite not PostgreSQL), or provisioning pipelines.	all	HIGH (reliability)	High — need test infrastructure
3	**MCP browser API key not generated** — `{{ mcp_browser_api_key }}` template variable in sysadmin docker-compose is never substituted by `env_setup.sh`. MCP browser runs unauthenticated.	ansible-runner	HIGH (security)	Low — add key generation to `env_setup.sh`

#	Item	Repo	Impact	Effort
4	**Root password unencrypted on disk** — provisioning job config.json sits in plaintext on Hub server's filesystem (JOBS_DIR) containing the root password for the target server.	hub	HIGH (security)	Medium — encrypt at rest, delete after job
5	**NextAuth beta dependency** — `next-auth@5.0.0-beta.30` in production. Breaking changes possible on upgrade.	hub	Medium (stability)	High — monitor for stable release, pin carefully
6	**Dual provisioning paths** — Both `ansible/runner.ts` (SSH inline) and `docker-spawner.ts` (container-based) exist, adding confusion about which path is canonical.	hub	Medium (maintainability)	Low — deprecate/remove one path

#	Item	Repo	Impact	Effort
7	**Legacy agent auth** — Both token-based and secret-hash-based auth maintained in parallel. Adds code complexity and potential security confusion.	orchestrator, agent	Medium (complexity)	Medium — migration plan + removal timeline
8	**No backup monitoring** — `backups.sh` writes `backup-status.json` but nothing reads it. Hub has no visibility into backup health.	ansible-runner, hub	Medium (operations)	Medium — add status endpoint + Hub polling
9	**Credential round-trip** — Hub generates creds → ansible-runner writes to server → agent reads and heartbeats back to Hub. Hub already has the creds.	hub, agent	Low (waste)	Low — eliminate re-sync, use Hub as source of truth
10	**No README in any repo** — all 5 repos lack README.md. Project documentation lives only in CLAUDE.md files (intended for AI assistants, not human developers).	all	Low (onboarding)	Low — create READMEs

11.2 5 Quick Wins

#	Win	Repo	Impact	Effort
1	**Add `mcp_browser_api_key` to `env_setup.sh`** — generate a random key and add the sed substitution. Fixes a real security hole.	ansible-runner	HIGH	~30 minutes
2	**Add auth to orchestrator playbook/tenant/task routes** — apply existing `AdminAuthDep` or `DashboardAuthDep` (already coded) to unprotected routes.	orchestrator	HIGH	~2 hours
3	**Clean up job config files after provisioning** — add a post-completion step in `jobService` to delete the config.json file containing the root password.	hub	HIGH	~1 hour

#	Win	Repo	Impact	Effort
4	**Wire `DashboardAuthDep` to playbook routes** — the dependency is already built and tested, just needs to be added to route decorators.	orchestrator	Medium	~1 hour
5	**Remove or deprecate `ansible/runner.ts`** — if docker-spawner is the canonical path, remove the SSH inline path to reduce confusion and attack surface.	hub	Medium	~2 hours

---

12. 10. Recommended Architecture

12.1 10.1 Repository Disposition

Repo	Recommendation	Rationale
letsbe-hub	**KEEP** — this is the central control plane	Largest, most feature-complete repo. Admin UI, customer management, order lifecycle, payment integration, server monitoring.

Repo	Recommendation	Rationale
letsbe-orchestrator	**KEEP** — essential per-tenant control plane	Runs on each tenant server. Task queue, agent management, playbook dispatch. Clean separation of concerns.
letsbe-sysadmin-agent	**KEEP** — essential per-tenant worker	Executes tasks from orchestrator. Well-designed executor pattern. Playwright scenarios working.
letsbe-ansible-runner	**KEEP but RENAME** to `letsbe-provisioner`	The name is misleading — it doesn't use Ansible at all. It's a Bash-based Docker provisioning container. The name should reflect what it actually does.
letsbe-mcp-browser	**EVALUATE for MERGE** into sysadmin-agent	The sysadmin agent already has embedded Playwright. The MCP browser is a separate HTTP API for the same capability. Currently unused. Consider either: (a) merging MCP browser functionality into the agent as an internal module, or (b) completing the integration and migrating all Playwright scenarios to use MCP browser as the browser backend. Option (b) is better long-term for LLM-driven exploratory browsing.

12.2 10.2 Missing Services / Components

Component	Purpose	Priority
Customer Portal	Self-service dashboard for customers to view server status, manage tools, see credentials. Built as pages under the existing Hub (Next.js app, `/customer` route group).	HIGH — needed for SaaS launch

Component	Purpose	Priority
DNS Automation Service	Integration with DNS provider (Cloudflare, Entri, or Route53) to automatically create/manage A records. Could be a service module within the Hub.	HIGH — blocks fully automated provisioning
Notification Service	Transactional email service for provisioning status updates, downtime alerts, cert renewal warnings. Extend existing `emailService` with event-driven triggers.	Medium
Backup Monitor	Agent-side executor that reads `backup-status.json` and reports to orchestrator. Hub-side dashboard for backup health.	Medium
Centralized Logging	Aggregated log collection across tenant servers. Consider Loki or similar lightweight log aggregation. Current approach of per-server logs doesn't scale.	Low (for initial launch)

12.3 10.3 Target Architecture for Production Launch

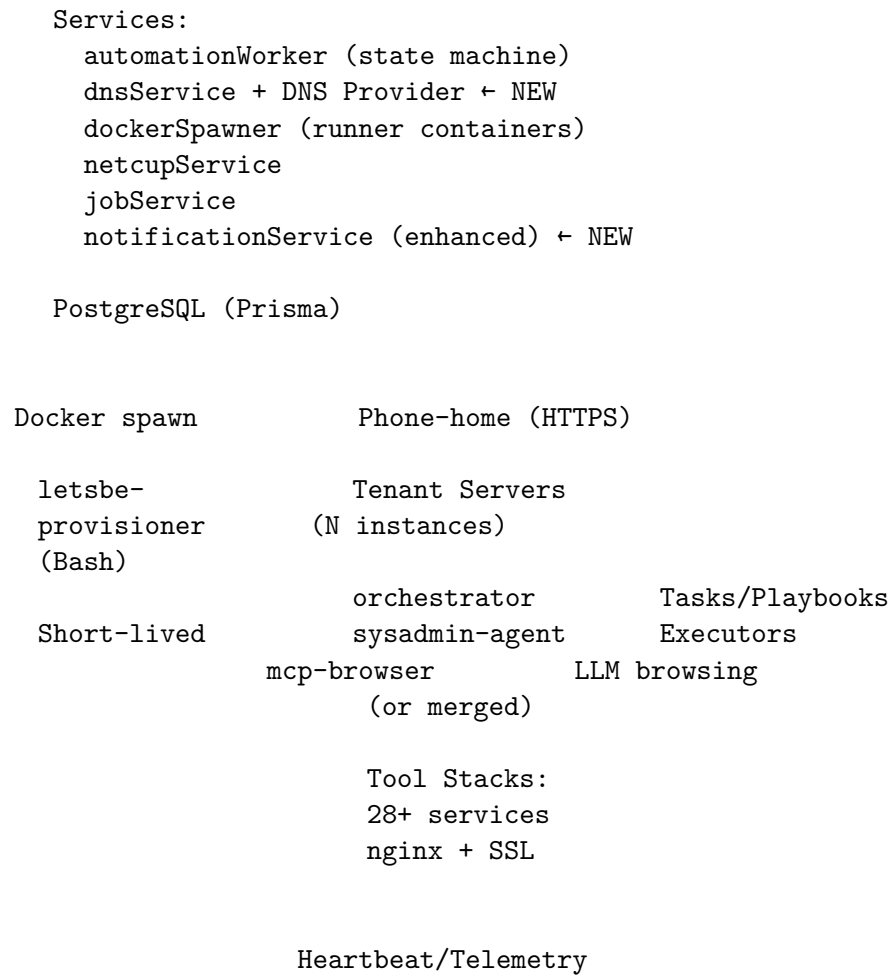
```
                    INTERNET

Stripe    Cloudflare/Entri    Netcup SCP



          letsbe-hub (Next.js)

Admin Portal (/admin)
Customer Portal (/customer)  ← NEW
Public API (/api/v1/public)
Orchestrator Phone-Home (/api/v1/orch)
Stripe Webhooks
```



Key changes from current state: 1. Add customer-facing portal within existing Hub 2. Add DNS provider integration for automated record management 3. Rename `ansible-runner` → `provisioner` to reduce confusion 4. Fix security gaps (auth on orchestrator routes, MCP browser API key, config cleanup) 5. Either merge MCP browser into agent or complete the integration 6. Remove legacy auth paths after migration period 7. Remove duplicate SSH provisioning path from Hub 8. Add integration tests for critical paths (provisioning, task execution, phone-home) 9. Enhance backup monitoring with agent-to-Hub reporting

---

End of Analysis