

Kernel R1 — Azure Deployment Pipeline

Evidence-based handover of the implemented Kernel scope. Business acceptance and release signatures remain open.

UAT package home · Editable UAT and technical registers

1. Deployment mechanism and release lineage

The implemented deployment pipeline is an **operator-driven Python/SSH pipeline to an Azure VM**. Its initial entry point is `MR33_MIGRATION_WORK/deployment/azure/pipeline.py`. Later Studio, Scheduler and UI corrections use release-specific `pack/rehearse/deploy/verify` scripts. No hosted Azure DevOps or GitHub Actions run is established by the retained deployment receipts. Azure hosting does not itself establish a hosted CI/CD pipeline.

The VM target recorded in the utility is `kernal@20.244.50.199`. The canonical user portal established on 11 October is `https://portal.kernelhealth.tech`; the compatible API origin remains `https://api.kernelhealth.tech`. The initial configuration's `app.kernelhealth.tech` belongs to an earlier deployment plan and must not overwrite the portal origin.

R1 is a handover baseline composed of multiple component releases. The 29 September base installation was followed by the portal edge release, MEDES technical delivery, 18 October Studio release, 19 October Scheduler/lifecycle release and later shared UI/IAM/Regulix corrections. Confirm the final active composition in the workbook's Release Baseline. Replaying an older package can remove later corrections even when that package passed its own tests.

This revision documents the implemented pipeline and recorded runs. It does not deploy, rebuild the application, rotate credentials, restore a database or rerun business regressions. A current environment freeze remains an operations/UAT entry gate.

Evidence: T01, T02, E05, E07, E01.

2. Deployment flow and stage gates

Freeze → Build/test → Clone rehearsal → Seal archive → Upload/stage → Backup/lock → Drain → Migrate → Activate runtime/configuration → Switch UI/edge → Verify → Issue UAT baseline.

D01 · Freeze and select release

Item	Operator instruction
Owner	Release owner
Procedure	Confirm change scope, active UI pointer/hash, service JARs, DB/config state, gateway routes and named target. Separate full installation from incremental changes.
Evidence / code	Release Baseline; DP01, DP18, DP22–23
Stop / recovery rule	Stop if expected baseline has changed; rebase/rehearse.

D02 · Build and test candidate

Item	Operator instruction
Owner	Engineering / QA
Procedure	Build the actual source candidate; retain typecheck/build logs, dependency locks, suite scripts, hashes and test environment. Historical capture pack does not build modules.
Evidence / code	E02, E08; DP02, DP08, DP15
Stop / recovery rule	Do not promote failed or untested artifact changes.

D03 · Rehearse migrations and rollback

Item	Operator instruction
Owner	DBA / configuration owner
Procedure	Restore protected snapshots into isolated databases; apply exact ordered inventory, replay, check data preservation and execute rollback. Rehearse business and configuration DB separately.
Evidence / code	T04, T20, T21; E08; DP12, DP16, DP20
Stop / recovery rule	Reject drift, non-replayable migrations or incomplete rollback.

D04 · Package and seal

Item	Operator instruction
Owner	Release engineer

Item	Operator instruction
Procedure	Create archive and per-file manifest. Bind readiness and acceptance to artifact/UI/runtime hashes. Keep private runtime archives outside the UAT documentation pack.
Evidence / code	DP08, DP09, DP15
Stop / recovery rule	Reject missing files or mismatched hashes.

D05 · Transfer and stage

Item	Operator instruction
Owner	Operations
Procedure	Upload over the established SSH connection; verify archive SHA256 on VM; extract into new /opt/kernel/releases/<release>; verify each manifest entry.
Evidence / code	DP01 upload/install; DP10; DP18
Stop / recovery rule	Leave active release unchanged on integrity failure.

D06 · Backup and guard active baseline

Item	Operator instruction
Owner	Operations / DBA
Procedure	Capture ACTIVE.json, private service registry, changed files/units, Boot snapshots and database undo metadata. Acquire activation lock and compare baseline hashes. Record DB backup and recovery target.
Evidence / code	DP11, DP18; /opt/kernel/backups/<release>
Stop / recovery rule	Stop on lock contention, drift or missing recovery evidence.

D07 · Quiesce affected producers

Item	Operator instruction
Owner	Operations / module owner
Procedure	For Studio release, stop Vista/Tenant producers and wait for existing render work to drain before stopping render workers. For Scheduler, prevent uncontrolled claims and track in-flight owner outcomes.
Evidence / code	DP11 drain loop; DP18; T11
Stop / recovery rule	Studio cancels if queued/rendering output cannot drain; never discard queue to force release.

D08 · Apply business then configuration changes

Item	Operator instruction
Owner	DBA / IAM / Axis owners

Item	Operator instruction
Procedure	Apply release-ordered transactions with time/lock bounds; register approved routes, mappings and service identity; preserve historical is_active nulls as rehearsed. Capture undo before/after state.
Evidence / code	T20–21; DP11–13, DP18–19
Stop / recovery rule	Business/config databases are separate commits; use recorded compensation if later phase fails.

D09 · Install and activate runtime

Item	Operator instruction
Owner	Operations
Procedure	Install hash-pinned worker/runtime artifacts and protected environment refs. Publish Boot configuration; wait for projection; start/restart only affected services and verify readiness.
Evidence / code	DP11, DP18; E05, E07
Stop / recovery rule	Restore prior runtime/config; wait for rollback projection; verify readiness.

D10 · Switch UI / gateway when in scope

Item	Operator instruction
Owner	Operations
Procedure	Atomically change generated UI ACTIVE.json only after prerequisites. Edge changes also validate Kong host routing and nginx -t, then reload Nginx. Preserve unrelated current sources.
Evidence / code	DP06, DP11, DP18, DP22–23
Stop / recovery rule	Restore matched pointer, route/origin and source/config backups.

D11 · Verify deployed installation

Item	Operator instruction
Owner	QA / Operations
Procedure	Check service readiness, public TLS/login, anonymous denials, allowed/denied profiles, current assets, worker heartbeat and scoped lifecycle checks. Record production mutation/delivery boundaries.
Evidence / code	E06, E07; T02, T19; E01
Stop / recovery rule	Reconcile failures; use component rollback or bounded fix. Health alone is not UAT.

D12 · Issue baseline and hand over

Item	Operator instruction
Owner	Release owner / UAT lead

Item	Operator instruction
Procedure	Record actual active component values, signatures, residual gaps, approved test accounts/data and source evidence. Run new business UAT in the agreed environment.
Evidence / code	Environment; Release Baseline; Entry Gates; 280 UAT cases
Stop / recovery rule	Keep acceptance Pending until exit gates and authorized signatures are complete.

3. Initial installation pipeline: exact stages

Run from the source workspace using the approved operator identity. The initial utility expects its private `latest.json`, a captured source backup inventory, SSH known-host file and protected release archive. These private assets are not included in the UAT ZIP. Establish the documented SSH control connection without storing passwords in commands or evidence.

`pack` captures current built runtime files/configuration and fresh database archives; it is not a clean Java/frontend build. Its private archive includes encrypted credential/provider material and document assets. Restrict that archive to operations; this UAT handover contains only sanitized evidence and references.

```
python3 MR33_MIGRATION_WORK/deployment/azure/pipeline.py pack
python3 MR33_MIGRATION_WORK/deployment/azure/pipeline.py upload
python3 MR33_MIGRATION_WORK/deployment/azure/pipeline.py install
python3 MR33_MIGRATION_WORK/deployment/azure/pipeline.py provision
python3 MR33_MIGRATION_WORK/deployment/azure/pipeline.py python_runtime
python3 MR33_MIGRATION_WORK/deployment/azure/pipeline.py restore
python3 MR33_MIGRATION_WORK/deployment/azure/pipeline.py configure
python3 MR33_MIGRATION_WORK/deployment/azure/pipeline.py start
python3 MR33_MIGRATION_WORK/deployment/azure/pipeline.py gateway
python3 MR33_MIGRATION_WORK/deployment/azure/pipeline.py verify
```

Stage group	Behavior and restriction
pack/upload/install	Capture archive → verify uploaded SHA256 → extract into a new release → verify per-file SHA256. Existing release directories must not be overwritten.
provision/python_runtime	Prepare host/runtime and pinned dependencies. Initial renderer dependency versions are historical; later Studio bundle pins supersede them.
restore	First installation into absent application databases only. Target code refuses existing databases; preserve the Kong database. Never use as an upgrade step.
configure/start	Configure private runtime paths and systemd. Boot starts before Config, then application modules; readiness gates startup. Workers/web follow dependencies.
gateway	Registers the original API-host Kong service/route. Portal route and origin setup are later DP06 work, not supplied by this old stage alone.
verify/status	Service activity, Java readiness, DB connectivity, public login and worker states. The historical worker check does not by itself prove fresh heartbeat or successful business output.

Stage group	Behavior and restriction
backup	Two PostgreSQL custom-format archives, catalogue validation and checksums. Off-VM copy, retention, encryption policy and restore RTO/RPO require operations evidence.
rollback	Switches previous base application release and restarts services. Database contents and later nested UI/worker changes require their component-specific recovery.
sync-tooling	Copies an explicit deployment-tool list and records hashes. Validate all listed files exist; do not use an old source tree to overwrite reviewed tooling.

4. Incremental release recipes and exact artifacts

Studio: Vista, Pulse, source/dataset and output workers

`build_plan.py` produces the ordered migration plan; `build_bundle.py` seals frontend, workers, source evidence and offline renderer wheel/font assets. The acceptance chain is migration rehearsal, renderer parity, packaged role/theme browser checks and native dataset/Pulse journeys. `readiness.py` binds those proofs to the archive hash. `deploy.py --rehearse-only` performs release preflight; `deploy.py` activates only a matching READY bundle. Both require the private prepared operator environment; these are not unauthenticated self-service commands.

Target `activate_remote.py` validates the archive/manifest, takes backups, stops Vista/Tenant producers, drains existing output and stops renderer workers. It applies business then config migrations, installs pinned workers, publishes MEDIFLEX_VISTA and MEDIFLEX_TENANT Boot snapshots, restarts services, verifies endpoints, switches the UI pointer and writes DEPLOYMENT.json. It restores scoped databases/configuration, files, roles, units and pointer on failure. This 18 October release explicitly excluded Scheduler and Java runtime replacement.

Scheduler and generic lifecycle

`build_plan.py` orders 89 business and 11 configuration migration entries. `package.py` requires the verified worker JAR, matching worker source hashes and built UI, then creates `release.tar` and `MANIFEST.json`. The target `deploy_remote.py <staged archive> <expected SHA256>` checks acceptance/rehearsal/profile/rollback proofs and current active hashes before proceeding. It is a release-specific target-side program, not a general deployment API.

The Scheduler runtime runs inside `kernel-mediflex_core.service` using the selected `Module_Runner.jar`. There is no separate enterprise-scheduler systemd service in this deployment recipe. The script creates a dedicated service identity, installs protected environment references before enabling Boot runtime configuration, waits for 20 registered endpoints, restarts

Core, requires a recent worker heartbeat, then switches the UI and checks profiles/lifecycle. The recorded release did not adopt existing jobs or attempt production delivery.

The release's `package.py` names a historical JAR in `/tmp`. Recreate/stage that exact verified artifact, or rebuild and issue a new manifest and acceptance record. Do not assume temporary files from an old run still exist. Application source, migrations and private artifacts are separate from this documentation ZIP.

Later UI / IAM / Regulix corrections

The scoped-query and Regulix-timeout deploy scripts update the currently active UI. Use their deployment receipts when freezing R1. A new candidate must include those corrections and preserve live runtime/configuration selections. Treat all module-specific scripts as dated recipes with preconditions, not an instruction to rerun every historical deployment in order.

Complete script/path/hash index · Recorded release and rollback locations.

5. Database and configuration promotion

The runtime database is `mediflex_db`; the deployment utility records configuration database `OPS-REF-a061a8c0be47` (resolve in operator source baseline). Verify names in the target environment before use. `OPS-REF-acaa995abe6a` (resolve in operator source baseline) and `endpoint_config` feed Boot configuration; business IAM mappings, Axis metadata and module functions/tables are applied through the business migration plan.

Migration order is the plan's array order, not alphabetical filename order. The enclosed Migration Inventory preserves all 36 Studio and 100 Scheduler entries, database group, ordinal and recorded SHA256. It also compares the current local file hash; a `SOURCE_CHANGED` or `SOURCE_NOT_PRESENT` entry prevents treating today's source as the old tested artifact. The two release inventories overlap in dependencies and must not be summed as 136 unique changes.

Business and configuration transactions do not form one cross-database atomic transaction. Stage executable undo and configuration predecessors before commit; on partial failure restore only the changes owned by that release. Boot publication/projection must be checked before service readiness is accepted. Preserve audit history and existing records; the Scheduler receipt preserves 35 historical `is_active` null rows while checking 256 tables.

Ordered migration inventory and source reconciliation; T20; T21.

6. Services, paths and ownership

Component	Recorded runtime / artifact	Operator verification
Base release	/opt/kernel/releases/<release>; /opt/kernel/current	Resolve active symlink and compare release receipt; do not assume base pointer identifies latest frontend.
UI release	OPS-REF-4c21a0c482a6 (resolve in operator source baseline); deployment/index.html and OPS-REF-4623114083d0 (resolve in operator source baseline) assets	Read active root/index hash and match dependent assets; keep later UI fixes.
Service registry	/opt/kernel/current/services.private.json	Authoritative executable/env selection, including web public origins; record redacted hashes, never secret values.
Persistent state	/opt/kernel/shared	Credential/document state survives application release changes; backup separately and preserve ownership.
Boot / Config	kernel-boot.service / kernel-config.service; loopback9090/8081	Dependency order and /health/ready; verify published version/projection.
Core / Scheduler	kernel-mediflex_core.service; loopback10047	Runtime JAR hash, scheduler.enabled, dedicated identity, recent heartbeat and enabled endpoints.
IAM / Axis	kernel-mediflex_iam.service10049; kernel-mediflex_axis.service10050	Native session and profile allow/deny; dynamic menus; control-level permissions.
Tenant / Vista	kernel-mediflex_tenant.service10048; kernel-mediflex_vista.service10051	Readiness and correct snapshot; source/dataset/template/report version behavior.
Browser/API host	kernel-web.service; HTTPS loopback11440	Correct portal origin, session gates, static asset access and routed APIs.
Preview / source worker	kernel-pulse-studio-preview.service; kernel-nexus-connections.service	Pinned executable/environment, worker health and owner result; source host allow-list.
Existing output workers	kernel-pulse-letter-rendering.service; kernel-pulse-email-output.service; kernel-vista-execution.service	Queue ownership, claim completion and output references; generation separate from delivery.
Edge	nginx and kong services	TLS and routes; public API denial checks; no public module/admin exposure.
Backups	/opt/kernel/backups/<release-or-time>	Matching ACTIVE/services/files/metadata/undo scripts and DB backup where applicable; verify off-VM protection.

Names/ports come from retained deployment source and receipts; confirm current installed units before operations. The initial25-service count is historical, not a current inventory.

7. Failure recovery and rollback decision

1. Stop further activation and preserve the failure receipt, phase, correlation IDs, active hashes and in-flight job/output identities. Identify whether only staging failed or a database/config/runtime/UI commit occurred.
2. Use the backup and undo files for the failed component release. A failed preflight leaves active application state alone; do not restore a whole database to undo a staging error.

3. For Scheduler failure after runtime selection, stop affected Core execution, restore configuration and wait for Boot rollback projection, apply release-owned business undo, restore private service selection/JAR, restart Core and verify readiness; restore UI pointer/web if it switched. Follow the script's state flags and generated rollback files.
4. For Studio failure, use its captured files/units/roles, database undo and Boot predecessor snapshots. Reconcile accepted output before restarting producers; return queued work to its existing owner path.
5. For edge failure, restore portal Nginx/site/origin/gateway/UI backup and only the route created by that change. Retain the established API route. Validate nginx -t before reload and test host/callback/session boundaries.
6. Verify actual allowed and denied profile behavior after recovery. Record the recovered build, configuration, queue/worker state and outstanding defects; issue a revised baseline before resuming UAT.

Rollback of code is not rollback of delivered emails, fax submissions, created third-party records or other accepted external effects. Reconcile owner receipts and provider status before retrying. The base `pipeline.py rollback` does not restore databases or all later nested component releases. A database restore requires its own approved recovery plan, loss window and reconciliation.

Current HA/failover, automated blue/green or canary promotion, off-VM retention, measured RTO/RPO and a hosted CI approval pipeline are not established by these receipts. Record responsible operations owners and evidence before asserting those capabilities.

8. Deployment evidence required at UAT entry

1. Record release/change ID, operator, approver, timestamps, target URL/VM and maintenance window.
2. Attach source/lockfile references, UI/runtime/archive hashes and matching regression proof hashes.
3. Attach ordered migration inventory, baseline schema/config hashes, rehearsal and rollback results.
4. Record current services, gateway route/service/plugin/version projection, TLS validity and private secret reference ownership.
5. Record backup location/checksum, off-VM copy, tested restore and rollback decision owner.
6. Record post-deploy health, current asset hashes, fresh worker heartbeats, expected anonymous denials and authenticated maker/checker allow/deny checks.

7. Link residual defects and acceptance restrictions. Complete Environment, Release Baseline, Entry Gates and Sign Off; historical PASS evidence does not populate new UAT results.

Kernel R1 UAT package • Prepared • Documentation and evidence consolidation only. No application deployment, access grant or business approval was performed by this packaging run.

Current release additions and acceptance

The following changes update the earlier handover baseline. Earlier results retain their original scope. The additional business UAT cases start Not run and require actual tester evidence.

Feature	Module	Change	Operating impact
F01	IAM	Explicit domain/profile mapping	Approved domain creation starts with zero profile links; pending mappings do not grant eligibility.
F02	Axis	Governed screen authoring	Create Screen, source/field definitions, access mapping, draft, preview and originating-record review.
F03	IAM / Axis	Reviews on the owning record	Review pending master, mapping, screen and control changes in their originating workspace.
F04	IAM	Named Super Administrator	Explicit scoped same-person review with administrator evidence; ordinary profiles retain independence checks.
F05	Axis	Screen and control metadata	Screen title/help and control label/help/format maintained through governed presentation changes.
F06	Axis	Owning screen controls	Controls list owning screens first; details and controls share context, refresh and Back behavior.
F07	Pulse / IAM	Notification outcome links	Published notification OPEN contract restored; recipient/context checks retained.
F08	MEDES	Activation qualification	Installed technical package; role grants, enabled APIs and review-route activation remain pending.
F09	Kernel	Documentation hub	Master index, separate manuals/demos/UAT, linked modules, current supplement and offline downloads.
F10	Axis	Consistent screen authoring and explicit header help	Shared record tabs, responsive form grid, Cancel/Save footer and multiline screen-header help stored in existing metadata.

Feature	Module	Change	Operating impact
F11	Axis	Bounded authorized editor options	Prepared authorization context eliminates repeated source checks; source and permission results remain unchanged.

[Feature register](#) · [Additional UAT cases](#) · [New evidence index](#)

MEDES entry gate: technical installation is complete; target activation is incomplete. Role grants, enabled APIs and document-review route must be verified before a live business demo. Provider-based delivery additionally requires approved provider setup and authorized destinations.

Verification boundary: the Axis screen-context release records 35 rollback-only database checks, 12 UI tests and 15 fixture-browser checks. The notification contract correction records 11 schema checks and 17 rollback-only database checks. These are not a complete authenticated production journey or a new client UAT sign-off.