Environment as Code für Murex MX.3.

Jede MX.3-Umgebung — PROD, DR, UAT, SIT, DEV, on-premises oder in der Cloud — wird in Git beschrieben statt von Hand konfiguriert. Darauf laufen EOD, Batch, Health-Checks und Housekeeping. Dasselbe Kommando, dasselbe Ergebnis, überall.

WAS IHR MUREX-TEAM DAVON HAT

- Kein stilles Auseinanderdriften mehr: Ein Kommando zeigt jede Einstellung, die sich zwischen PROD und UAT unterscheidet — bevor daraus ein Incident im Batch oder ein gescheitertes Release-Wochenende wird.
- Umgebungsarbeit in Minuten statt Tagen: Neue Umgebung, Refresh, Murex-Versionswechsel: ein paar Zeilen im Inventory ändern, rendern, anwenden. Das Wissen liegt im Repository, nicht im Kopf eines einzelnen Engineers.
- Nachweise statt Behauptungen: Konfiguration ist in Git versioniert, mit vollständiger Historie. Secrets bleiben in Ihrem Vault (CyberArk, Ansible Vault, Infisical) und werden erst beim Anwenden aufgelöst. Was in Git versioniert ist, lässt sich belegen; was ein Skript zur Laufzeit zusammensetzt, nicht.
- Tooling, das Sie nicht erst bauen müssen: Sie starten DevOps und CI/CD für Murex nicht mit einem leeren Playbook. MxENV3 ist eine Basis, die seit 1998 im Bankbetrieb läuft. Ihr Team baut eigene Playbooks und Pipelines darauf — statt Umgebungen, Startreihenfolgen und Checks noch einmal zu erfinden.

DER BETRIEB

- Ein Kommando, jede Umgebung: Ob PROD, DR oder die fünfte UAT: runprocessingscript, mxcheck, housekeeping oder start lesen alles Nötige aus dem Inventory der Umgebung. Keine umgebungsspezifischen Skripte, kein Copy-Paste zwischen Hosts, keine Überraschungen, wenn der DR-Standort wirklich gebraucht wird.
- Tausende EOD-Jobs, zuverlässig gesteuert: Jedes Processing-Skript, jedes Makro und jeder Ant-Task liefert ein konsolidiertes Ergebnis: Status, Laufzeit, aufgefangene Logs und die geparste Murex-Antwort. Der Scheduler verzweigt auf Fakten statt auf einen Shell-Exit-Code. Dieselbe Job-Definition läuft auf jeder Umgebung, weil Host, User, Pfad und Parameter aus dem Inventory kommen.
- Scheduler-Jobs aus einer Tabelle erzeugt: Die EOD-Kette wird einmal als Tabelle gepflegt. MxENV3 erzeugt daraus die AutoSys-Jobdefinitionen (JIL) mit den richtigen Hosts, Usern, Pfaden und Datumsbedingungen je Umgebung. Dieselbe Kette in PROD und UAT, ohne hunderte Jobdefinitionen einzeln zu pflegen; der Weg zurück von JIL in die Tabelle ist ebenso möglich. Weitere Scheduler richten wir auf Anfrage ein.
- EOD-Alarmierung an die eingetragene Bereitschaft: Bricht ein EOD-Job ab, sammelt MxENV3 den Log-Auszug des Jobs und schickt ein Servicedesk-Ticket per Mail, dazu optional eine SMS über das Mail-Gateway. Wer Bereitschaft hat, trägt sich vorher im Bereitschaftsplan ein; das Ticket geht an diese Person. Transport und Textbausteine kommen aus dem Inventory.

IM WERKZEUGKASTEN

- Health-Checks: mxcheck prüft Fileserver, Middleware, User-Sessions und offene Dateien; jeder Service muss antworten, verglichen mit einem Soll-Snapshot. Status und Metriken kommen als JSON oder Text, mit Exit-Codes für den Scheduler und Schwellwerten aus dem Inventory.
- Oracle-SQL gegen die richtige Datenbank: runsql führt Queries, DML, DDL und PL/SQL per sqlplus gegen financial, datamart, MLC oder VaR aus. Die Verbindung kommt aus dem Inventory, das Passwort aus Ihrem Vault. Dazu Dry-Run, CSV/JSON-Ausgabe und Fail-fast-Batches.

Business Technology Consulting OÜ · Sepapaja tn 6, 15551 Tallinn, Estonia · info@mxenv3.com · https://mxenv3.com

- Murex-Jobs mit einem konsolidierten Ergebnis: runant, runmacro und runprocessingscript rufen Murex-Skripte auf und liefern Kommando, Host, Laufzeit, Logs und die geparste Murex-Antwort als ein JSON — für den Scheduler und als Nachweis.
- Housekeeping und Archivierung: Altersbasierte Archivierung in verifizierte tar.gz und anschließende Bereinigung, gesteuert über Presets im Inventory: Logs, GC-Logs, Core-Dumps, datierte Exporte. Erst Dry-Run, dann auf einem Host oder allen.
- Diagnose: Business-Datum gegen heute prüfen, JVM-Heap pro Host mit Schwellwerten, MXML-Tasks und Middleware-Ping gegen einen git-versionierten Soll-Snapshot. Dazu Purge über eine Node-Allow-List und Ad-hoc-Kommandos auf eine ganze Rolle.
- Zentraler Inventory-Server: Ein optionaler Server pro Domäne (TLS und Token) liefert das git-versionierte Inventory an schlanke Clients. Er synchronisiert versionsspezifische Templates per Prüfsumme und kann Jobs stellvertretend ausführen. Clients aktualisieren sich von ihm selbst.

SO FUNKTIONIERT ES

- 01 Umgebungen beschreiben: Jede Umgebung ist ein kleines Inventory im Standard-Ansible-Format. Server sind nach Murex-Rolle gruppiert (Session, Processing, XML-Server …), dazu die Einstellungen, die von den bankweiten Defaults abweichen. Das Format kennt Ihr Team bereits.
- 02 Werte auflösen und vergleichen: Defaults, Umgebungs-, Rollen- und Host-Einstellungen werden zur wirksamen Konfiguration zusammengeführt. Einen Wert abfragen, zwei Umgebungen vergleichen, die volle Matrix rendern — immer mit klarer Antwort, welcher Wert wo gilt.
- 03 Planen, anwenden, betreiben: Konfigurationsdateien werden für die Murex-Version der Umgebung gerendert und ins Anwendungsverzeichnis übernommen. EOD-Schritte, Checks, Housekeeping und Service-Orchestrierung laufen aus demselben Inventory. Standardmäßig Dry-Run, Ausführung auf Wunsch, mit Exit-Codes, auf die Ihr Scheduler verzweigt.

GRENZEN

MxENV3 patcht, installiert und aktualisiert MX.3 nicht selbst, berührt keine Geschäfts- oder Marktdaten und ersetzt weder die herstellerseitigen Installations- und Administrationswerkzeuge noch Ihren Scheduler, Ihr Monitoring oder Ihre CI/CD. Es verwaltet das Drumherum: Umgebungskonfiguration, Service-Orchestrierung, Batch-Ausführung und die Betriebsroutine über 30 bis 50 Umgebungen. Auf einer von uns noch nicht validierten Murex-Version bricht das Anwenden der Konfiguration mit --strict ab, statt zu raten.

HERKUNFT

Das Rewrite von 2026 trägt dasselbe Betriebswissen über jede Generation von MxGeneration 2000 bis MX.3 — auf neuem Fundament: ein in Go kompiliertes Binary, Standard-Inventory, Vault-Anbindung, Produktionsschutz. Wo früher ein Perl-Baum mit rund 250 Modulen auf jedem Murex-Host installiert und über Jahre gepflegt werden musste, liegt heute eine einzige Datei. Keine Interpreter-Version, keine Abhängigkeiten, kein Compiler auf dem Server. Kopieren und ausführen. Ein Aufruf kostet Millisekunden CPU-Zeit statt einer halben Sekunde und weniger als die Hälfte des Speichers; das Rendern braucht pro Konfigurationsdatei rund 30× weniger CPU.

Gemessen auf identischer Hardware gegen das Perl-Vorgängertool, zehn Läufe je Fall: Kommandostart 4 bis 5 ms CPU und ~24 MB statt ~520 ms und ~54 MB; applyconfig mit 880 Dateien ~60 ms und ~40 MB statt ~2,2 s und ~89 MB. Die Werte gelten für das ausgelieferte Produktions-Binary. Start und Stop von Services sind nicht Teil der Messung.

NÄCHSTER SCHRITT

Business Technology Consulting OÜ · Sepapaja tn 6, 15551 Tallinn, Estonia · info@mxenv3.com · https://mxenv3.com

Schicken Sie uns einen anonymisierten Überblick Ihrer Murex-Umgebungen, oder auch nur Anzahl und Versionen. In 45 Minuten zeigen wir Ihnen das resultierende Inventory, die Konfigurationsmatrix und einen Startplan. Kein Zugriff auf Ihre Systeme nötig. Danach können Sie auf einer Nicht-Produktionsumgebung selbst testen: Die Evaluierung beginnt mit ausschließlich lesenden Kommandos — Konfigurationsmatrix, Umgebungsvergleich, Drift-Abgleich gegen die ausgelieferten Dateien. Geschrieben wird erst, wenn Sie es freigeben.

Lizenz	Jahreslizenz pro Institut, skaliert mit der Zahl der gleichzeitig laufenden Umgebungen.
Kontakt	info@mxenv3.com · https://mxenv3.com

Business Technology Consulting OÜ · Sepapaja tn 6, 15551 Tallinn, Estonia · info@mxenv3.com · https://mxenv3.com