Cloud-Automation → Helm

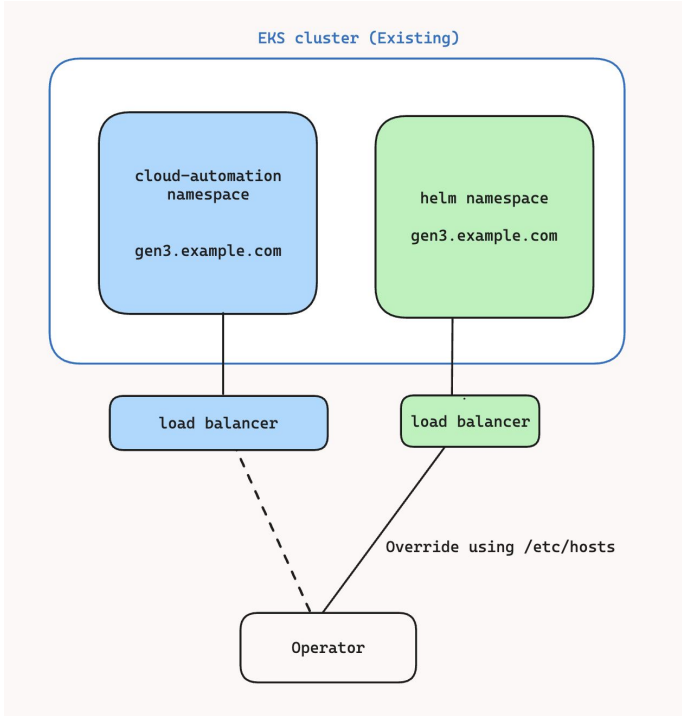
Motivation - Why migrate from cloud-automation to Helm?

- **Security Enhancement**
  - Eliminate plaintext secrets stored in "adminvms" - current approach exposes sensitive data in version control and local environments
- **Advanced Secrets Management**
  - Leverage AWS Secrets Manager and External Secrets Operator for automated, secure credential handling
- **Industry-Standard GitOps**
  - Adopt declarative infrastructure management following cloud-native best practices used across the industry
- **Infrastructure Separation**
  - Decouple infrastructure provisioning from application deployments for better maintainability and security. Easier to deploy Gen3 to other cloud-providers.

High Level Migration Strategy

Blue/Green Deployment Approach

- **Zero-Downtime Migration**: Run both environments simultaneously during transition
- **Namespace Isolation**: Deploy Helm environment in separate namespace to avoid conflicts
- **Traffic Management**: Use `/etc/hosts` file modifications to point to new load balancer initially
- **Pod-Level Routing**: We deploy a MutatingWebhook to inject host aliases into all pods in namespaces with “helm”, enabling gradual service-by-service migration without DNS cutover.



Prerequisites

- There are the prerequisites you need to have set up in order to do the migration.

ArgoCD

GitOps

Link

External Secrets Operator

Secrets

Link

Terraform + Terragrunt

Infrastructure Management

+ Atlantis

Migration Steps

Step 1 - Migrate Secrets & Generate Configs

Tool: migrate-to-helm.py script

Automated Process:

- **Secret Extraction**:
  - Parse existing cloud-automation configurations to identify all secrets
- **AWS Secrets Manager Population**:
  - Automatically create corresponding secrets in AWS Secrets Manager
- **Values Generation**:
  - Produce clean `values.yaml` files with service configurations but no embedded secrets
- **Validation**:
  - Ensure all configuration parameters are properly mapped to new Helm structure

Migration Steps

Step 2 - Setup IAM IRSA

(IAM roles for service accounts)

Example: gen3-terragrunt (private repo)

Reference: gen3-terraform IRSA roles (open source)
Docs: aws docs

Purpose: Giving cloud-identities to services

Migration Steps

Step 3 - GitOps repository

Template Repository: gen3-gitops

Environment Structure: Organized configuration per environment with ArgoCD Applications

Configure ArgoCD

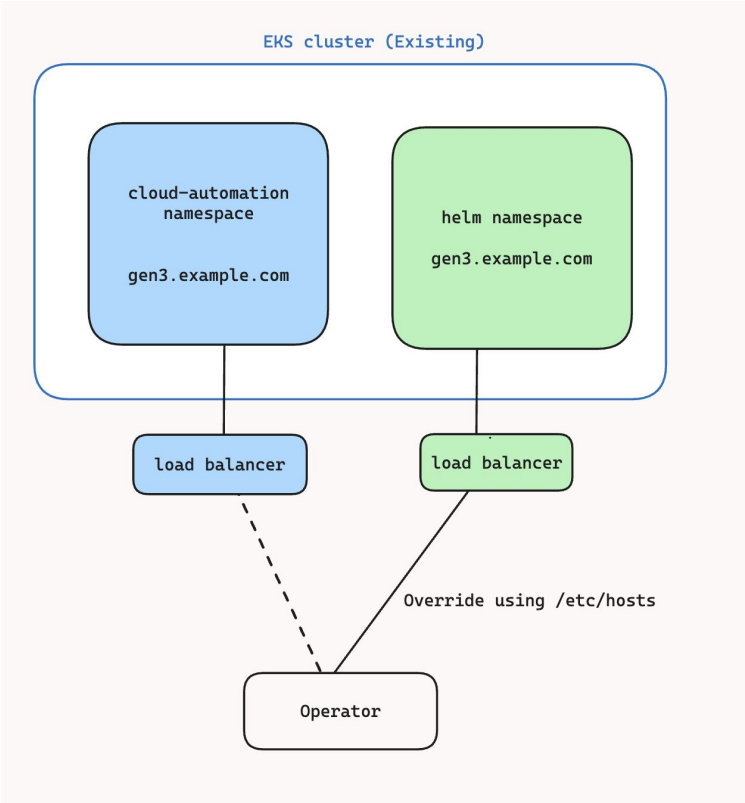
Migration Steps

Step 4 - Validation & Testing

Functionality Testing: Comprehensive manual testing to ensure feature parity between old and new environments

Script Limitations: Current migration script handles basic cases - enhancements added iteratively based on real migration experiences. You must validate the completeness and correctness of the output for your context.

Regression Prevention: Document and test all service interactions before cutover



AdminVM → Generic Jumpbox

- How to access production?
- AL23 FIPS based “JumpBox”
  - Kubectl
  - PSQL
  - ES
  - AWS cli
- Access using “aws ssm” - for auditability
- No secrets

Future Improvements Plan

- CSOC Portal Updates
- Crossplane
- GitOps improvements
  - Remove large files out of values.yaml
  - Automatic updates
    - Slack bots
  - Incorporate release process into gen3-helm
    - Releases like 2025.06
- CI testing
  - Ephemeral testing environments
  - Run entire gen3 test suite on a values.yaml