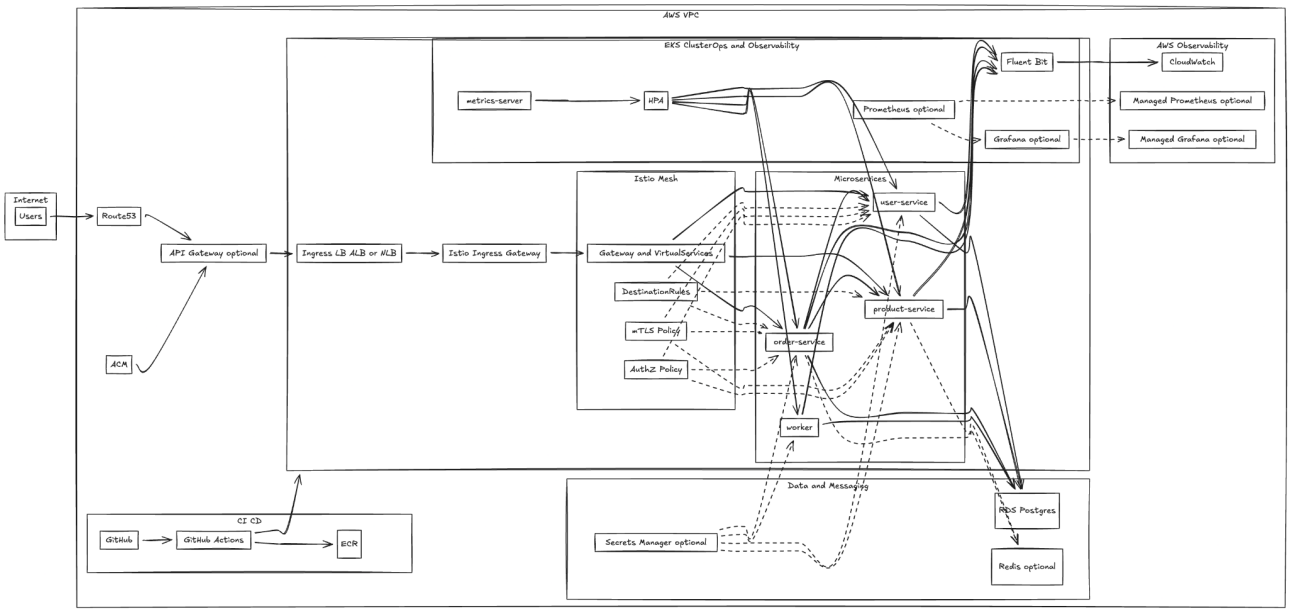
Sheshir-Runbook

I use this runbook to provision, deploy, and operate the Toy Production microservices system on AWS using EKS, RDS Postgres, ElastiCache Redis, ALB Ingress, External Secrets Operator, and GitHub Actions OIDC. All steps below are written so I can execute them end-to-end without detours.

Reference architecture



Scope

I deploy three Node.js microservices (user-service, product-service, order-service) and two backing services (RDS Postgres, ElastiCache Redis). I expose the APIs externally through an AWS Application Load Balancer (ALB) created by the AWS Load Balancer Controller. I keep the data layer private. I do not commit secrets into Git. I rely on IAM roles (IRSA) and GitHub Actions OIDC.

Repository assumptions

I assume my repo contains:

- `assessment/` – application code (services/) and helper scripts (`scripts/acceptance.sh`, `scripts/init.sql`, etc.).
- `deploy/helm/toy-production/` – Helm chart for the platform deployment (Deployments, Services, Ingress, DB init job, config).
- `infra/terraform/envs/dev/` – Terraform code for VPC, EKS, ECR, RDS, Redis, IAM roles/policies, and outputs consumed by Helm/CI.

One-go deployment procedure

I follow this order every time. If I run it from a clean AWS account and a clean cluster, it completes without manual fixes.

0) Local tooling and AWS access

I install and configure these tools on my workstation (or a CI runner) before I start:

- AWS CLI v2 (authenticated to the target AWS account).
- Terraform >= 1.5.
- kubectl compatible with my EKS version.
- Helm v3.
- eksctl (used once to map the GitHub Actions role to EKS RBAC).
- Docker (only needed if I build/push images from my workstation; CI can do it too).

```
# I set these once per shell session
export AWS_REGION="us-east-1"
export AWS_ACCOUNT_ID="222066942466"

export CLUSTER_NAME="toy-production-dev-eks"
export K8S_NAMESPACE="toy-production"
export HELM_RELEASE="toy-production"

# I keep a short name prefix for consistent tagging
export NAME_PREFIX="toy-production-dev"
```

1) Provision AWS infrastructure with Terraform

I use Terraform as the source of truth. I do not click-create core resources in the console.

I provision (minimum):

- VPC with public + private subnets across at least two AZs, plus NAT for private egress.
- EKS cluster and a managed node group in private subnets.
- ECR repositories: one per service (user-service, product-service, order-service).
- RDS Postgres in private subnets (not publicly accessible).
- ElastiCache Redis in private subnets.
- IAM: IRSA roles for AWS Load Balancer Controller and External Secrets Operator; GitHub Actions OIDC role for CI/CD.
- Secrets Manager secret holding the DB connection string (key: `database_url`).

```
cd infra/terraform/envs/dev
terraform init
terraform apply -auto-approve

# I capture the outputs I will reuse:
terraform output
```

I record these Terraform outputs (names are illustrative; I align to my actual module outputs):

Output	Purpose
eks_cluster_name	EKS cluster name used by kubectl and CI.
ecr_user_service_repo_url	ECR repo URL for user-service image.
ecr_product_service_repo_url	ECR repo URL for product-service image.
ecr_order_service_repo_url	ECR repo URL for order-service image.
rds_endpoint	RDS hostname (private).
redis_endpoint	Redis endpoint (private).
db_secret_name	AWS Secrets Manager secret name storing `database_url`.
alb_controller_irsa_role_arn	IRSA role ARN for AWS Load Balancer Controller.
external_secrets_irsa_role_arn	IRSA role ARN for External Secrets Operator.
github_actions_role_arn	Role assumed by GitHub Actions via OIDC.

2) Configure cluster access

After Terraform finishes, I configure my kubeconfig to point to the new cluster.

```
aws eks update-kubeconfig --region "${AWS_REGION}" --name "${CLUSTER_NAME}"
kubectl get nodes -o wide
```

3) Install EKS add-ons in the correct order

I install cluster add-ons before I deploy application workloads, because a broken admission webhook or missing CRD can block unrelated resources.

3.1) metrics-server (required for HPA)

```
helm repo add metrics-server https://kubernetes-sigs.github.io/metrics-server/
helm repo update

helm upgrade --install metrics-server metrics-server/metrics-server   -n kube-system   --set args={-

kubectl get deployment -n kube-system metrics-server
```

3.2) AWS Load Balancer Controller (ALB Ingress)

I install the AWS Load Balancer Controller before I apply any Ingress. I use IRSA so I do not store AWS keys in the cluster.

```
helm repo add eks https://aws.github.io/eks-charts
helm repo update

helm upgrade --install aws-load-balancer-controller eks/aws-load-balancer-controller   -n kube-syste

kubectl get pods -n kube-system -l app.kubernetes.io/name=aws-load-balancer-controller -o wide
kubectl get endpoints -n kube-system aws-load-balancer-webhook-service -o wide
```

I only continue after the webhook service shows non-empty endpoints.

3.3) External Secrets Operator (ESO)

I pin ESO to a compatible version and install CRDs via Helm.

```
helm repo add external-secrets https://charts.external-secrets.io
helm repo update

kubectl create namespace external-secrets --dry-run=client -o yaml | kubectl apply -f -

helm upgrade --install external-secrets external-secrets/external-secrets   -n external-secrets   --

kubectl get pods -n external-secrets
```

4) Prepare secrets and TLS for RDS Postgres

I store a single `database_url` string in Secrets Manager and I sync it into Kubernetes with ESO as secret `toy-production-db`.

```
# I URL-encode DB passwords containing special characters.
export DBPW='exampleP@ssw0rd%()|'
python3 - <<'PY'
import os, urllib.parse
print(urllib.parse.quote(os.environ["DBPW"], safe=""))
PY
unset DBPW
```

I build the connection string using the encoded password and I force SSL.

```
postgres://toyapp:<URLENCODED_PASSWORD>@<rds_endpoint>:5432/toy_production?sslmode=verify-full
```

I write this value into Secrets Manager under the JSON key `database_url`.

```
aws secretsmanager put-secret-value   --region "${AWS_REGION}"   --secret-id "<terraform_output_db_s
```

I also provide the RDS CA bundle to pods so Node can validate certificates.

```
kubectl -n "${K8S_NAMESPACE}" create configmap rds-ca-bundle   --from-file=rds-combined-ca-bundle.pe
```

5) Configure External Secrets (ClusterSecretStore + ExternalSecret)

I create the namespace first, then the ClusterSecretStore, then the ExternalSecret, and I wait until the Kubernetes Secret exists before I install the application chart.

```
kubectl create namespace "${K8S_NAMESPACE}" --dry-run=client -o yaml | kubectl apply -f -

kubectl apply -f deploy/k8s/external-secrets/clustersecretstore-aws-secretsmanager.yaml
kubectl -n "${K8S_NAMESPACE}" apply -f deploy/k8s/external-secrets/toy-production-db.externalsecret.

kubectl -n "${K8S_NAMESPACE}" wait --for=condition=Ready externalsecret/toy-production-db --timeout=
kubectl -n "${K8S_NAMESPACE}" get secret toy-production-db
```

6) Build and push service images to ECR

I tag images with an immutable identifier (Git SHA). I do not rely on `latest`.

```
export IMAGE_TAG="$(git rev-parse HEAD)"

aws ecr get-login-password --region "${AWS_REGION}"   | docker login --username AWS --password-stdin

USER_REPO="<terraform_output_ecr_user_service_repo_url>"
```

```
PRODUCT_REPO="<terraform_output_ecr_product_service_repo_url>"
ORDER_REPO="<terraform_output_ecr_order_service_repo_url>"

cd assessment
docker build -f services/user-service/Dockerfile -t "${USER_REPO}:${IMAGE_TAG}" .
docker build -f services/product-service/Dockerfile -t "${PRODUCT_REPO}:${IMAGE_TAG}" .
docker build -f services/order-service/Dockerfile -t "${ORDER_REPO}:${IMAGE_TAG}" .

docker push "${USER_REPO}:${IMAGE_TAG}"
docker push "${PRODUCT_REPO}:${IMAGE_TAG}"
docker push "${ORDER_REPO}:${IMAGE_TAG}"
cd ..
```

7) Deploy the application Helm chart (Ingress last)

I avoid Ingress races by deploying core workloads first, then enabling Ingress.

```
helm upgrade --install "${HELM_RELEASE}" deploy/helm/toy-production   -n "${K8S_NAMESPACE}"   -f dep

helm upgrade "${HELM_RELEASE}" deploy/helm/toy-production   -n "${K8S_NAMESPACE}"   -f deploy/helm/t
```

8) Verify and run acceptance

```
kubectl rollout status deployment -n "${K8S_NAMESPACE}" user-service --timeout=5m
kubectl rollout status deployment -n "${K8S_NAMESPACE}" product-service --timeout=5m
kubectl rollout status deployment -n "${K8S_NAMESPACE}" order-service --timeout=5m

ALB_HOSTNAME=$(kubectl get ingress -n "${K8S_NAMESPACE}" "${HELM_RELEASE}"   -o jsonpath='{.status.l

BASE_URL="http://${ALB_HOSTNAME}" ./assessment/scripts/acceptance.sh
```

One-go bootstrap script

When I want to run the full setup from scratch in one shot, I create a script like the one below and execute it from repo root. I only run it after I verify my Terraform backend configuration is correct.

```
#!/usr/bin/env bash
set -euo pipefail

export AWS_REGION="us-east-1"
export AWS_ACCOUNT_ID="222066942466"
export CLUSTER_NAME="toy-production-dev-eks"
export K8S_NAMESPACE="toy-production"
export HELM_RELEASE="toy-production"
export IMAGE_TAG="$(git rev-parse HEAD)"

# 1) Terraform
pushd infra/terraform/envs/dev
terraform init
terraform apply -auto-approve
popd

# 2) kubeconfig
aws eks update-kubeconfig --region "${AWS_REGION}" --name "${CLUSTER_NAME}"
kubectl get nodes

# 3) Add-ons
helm repo add eks https://aws.github.io/eks-charts
```

```
helm repo add external-secrets https://charts.external-secrets.io
helm repo add metrics-server https://kubernetes-sigs.github.io/metrics-server/
helm repo update

helm upgrade --install metrics-server metrics-server/metrics-server   -n kube-system --set args={--k

helm upgrade --install aws-load-balancer-controller eks/aws-load-balancer-controller   -n kube-syste

kubectl get endpoints -n kube-system aws-load-balancer-webhook-service -o wide

helm upgrade --install external-secrets external-secrets/external-secrets   -n external-secrets --cr

# 4) Namespace and secrets sync
kubectl create namespace "${K8S_NAMESPACE}" --dry-run=client -o yaml | kubectl apply -f -
kubectl apply -f deploy/k8s/external-secrets/clustersecretstore-aws-secretsmanager.yaml
kubectl -n "${K8S_NAMESPACE}" apply -f deploy/k8s/external-secrets/toy-production-db.externalsecret.y
kubectl -n "${K8S_NAMESPACE}" wait --for=condition=Ready externalsecret/toy-production-db --timeout=5

# 5) Build & push
aws ecr get-login-password --region "${AWS_REGION}"   | docker login --username AWS --password-stdin

USER_REPO="$(terraform -chdir=infra/terraform/envs/dev output -raw ecr_user_service_repo_url)"
PRODUCT_REPO="$(terraform -chdir=infra/terraform/envs/dev output -raw ecr_product_service_repo_url)"
ORDER_REPO="$(terraform -chdir=infra/terraform/envs/dev output -raw ecr_order_service_repo_url)"

pushd assessment
docker build -f services/user-service/Dockerfile -t "${USER_REPO}:${IMAGE_TAG}" .
docker build -f services/product-service/Dockerfile -t "${PRODUCT_REPO}:${IMAGE_TAG}" .
docker build -f services/order-service/Dockerfile -t "${ORDER_REPO}:${IMAGE_TAG}" .
docker push "${USER_REPO}:${IMAGE_TAG}"
docker push "${PRODUCT_REPO}:${IMAGE_TAG}"
docker push "${ORDER_REPO}:${IMAGE_TAG}"
popd

# 6) App deploy (Ingress last)
helm upgrade --install "${HELM_RELEASE}" deploy/helm/toy-production   -n "${K8S_NAMESPACE}" -f deploy

helm upgrade "${HELM_RELEASE}" deploy/helm/toy-production   -n "${K8S_NAMESPACE}" -f deploy/helm/toy

# 7) Acceptance
ALB_HOSTNAME=$(kubectl get ingress -n "${K8S_NAMESPACE}" "${HELM_RELEASE}" -o jsonpath='{.status.loa
BASE_URL="http://${ALB_HOSTNAME}" ./assessment/scripts/acceptance.sh
```

CI/CD (GitHub Actions OIDC) one-go setup

I configure CI so every push to `main` builds images, pushes to ECR, and deploys the Helm chart to EKS.

9) IAM: GitHub OIDC role trust and EKS RBAC mapping

I create the GitHub OIDC provider and IAM role in Terraform. I ensure the trust policy `sub` matches the repo and branch.

```
# sub must match:
# repo:<OWNER>/<REPO>:ref:refs/heads/main
```

After Terraform, I map the GitHub Actions role into EKS RBAC once.

```
eksctl create iamidentitymapping   --cluster "${CLUSTER_NAME}"   --region "${AWS_REGION}"   --arn "<
```

10) GitHub repository configuration

In GitHub, I set:

- Repository secret: `AWS_ROLE_TO_ASSUME` = ``.
- Repository variables: `AWS_REGION`, `EKS_CLUSTER_NAME`, `HELM_RELEASE`, `K8S_NAMESPACE`.

11) Workflow (build-and-deploy)

I keep my workflow deploy steps aligned with the two-phase Helm install.

```
# I store the workflow under .github/workflows/build-and-deploy.yaml
# (I keep the same content as the workflow skeleton in this runbook).
```

Rollback

When I need to roll back, I use Helm release history.

```
helm history "${HELM_RELEASE}" -n "${K8S_NAMESPACE}"
helm rollback "${HELM_RELEASE}" <REVISION> -n "${K8S_NAMESPACE}" --wait --timeout 15m
```

Operational checks I run first during incidents

- I confirm the Ingress has an ALB hostname and the ALB target groups are healthy.
- I check pod readiness and restarts: `kubectl get pods -n toy-production`.
- I check logs for DB/Redis errors: `kubectl logs -n toy-production deploy/order-service --tail=200`.
- I validate External Secrets sync: `kubectl describe externalsecret -n toy-production toy-production-db`.

troubleshoot

This chapter lists only the troubleshooting actions that were effective for me and the purpose of each action.

1. External Secrets CRD install failure (.spec.versions[0].selectableFields schema error)

I pinned External Secrets Operator to version 0.9.11 and enabled CRD installation (`--set installCRDs=true`). Purpose: make the CRDs apply cleanly on my Kubernetes version.

2. ESO install blocked by AWS Load Balancer webhook with no endpoints

I found `aws-load-balancer-webhook-service` had no endpoints and removed stale webhook configurations/services, then I reinstalled AWS Load Balancer Controller. Purpose: unblock API writes intercepted by a broken admission webhook.

3. AWS Load Balancer Controller pods not created (serviceaccount not found)

I ensured the `aws-load-balancer-controller` service account existed and the Helm release referenced it correctly. Purpose: allow the controller to start and serve the webhook.

4. Helm YAML/template parse errors (get-docker.sh and malformed ConfigMap YAML)

I removed non-Kubernetes scripts from Helm templates and fixed YAML indentation/keys in the DB init ConfigMap. Purpose: allow Helm to render valid manifests.

5. ExternalSecret schema error (.data field not declared in schema)

I corrected my ExternalSecret manifest to match the installed ESO API version and fields. Purpose: allow ESO to reconcile and create the Kubernetes Secret.

6. Pods failing because secret toy-production-db not found

I waited for ESO to sync and verified the Secret existed before restarting workloads. Purpose: prevent init containers from failing on missing secrets.

7. Database URL percent-encoding failures (invalid percent-encoded token)

I URL-encoded the RDS password and stored the encoded `database_url` in Secrets Manager. Purpose: make the connection string parseable by clients.

8. Readiness failing with 'no pg_hba.conf entry ... no encryption'

I appended SSL settings to the DB connection string (`sslmode=require`). Purpose: enforce encrypted connections to RDS and satisfy readiness checks.

9. GitHub Actions OIDC assume-role unauthorized

I corrected the IAM role trust policy `sub` format. Purpose: allow CI to assume the role via OIDC.

10. GitHub Actions could assume role but kubectl failed (client asked to provide credentials)

I mapped the GitHub Actions IAM role into EKS RBAC. Purpose: authorize CI to the Kubernetes API.

11. Helm namespace ownership/import conflicts

I removed Namespace creation from the chart and created the namespace outside Helm. Purpose: avoid Helm ownership metadata collisions.

12. ECR image pull failures (tag not found)

I set `global.imageTag` to an existing immutable tag in ECR instead of relying on `latest`. Purpose: prevent ImagePullBackOff and make deployments deterministic.