Migrating large-scale self-managed Kubernetes clusters to Amazon EKS

AWS Prescriptive Guidance

AWS Prescriptive Guidance: Migrating large-scale self-managed Kubernetes clusters to Amazon EKS

Table of Contents

Migrating large-scale self-managed Kubernetes clusters to Amazon EKS

Pradip kumar Pandey and Pratap Kumar Nanda, Amazon Web Services

Overview

Moving from a self-managed Kubernetes setup toAmazon Elastic Kubernetes Service (Amazon EKS) represents a significant shift in how organizations handle container orchestration. This transition involves moving your existing workloads, configurations, and operational practices from clusters you manage yourself to a managed Kubernetes service on AWS.

This guide helps you migrate self-managed Kubernetes to Amazon EKS by reducing risks, saving time and cost, and providing clear planning frameworks. It builds team confidence, supports informed decision-making, and enables effective stakeholder communication. The guide serves as a practical reference throughout the migration journey, helping organizations avoid costly mistakes while transitioning from infrastructure maintenance to focusing on application development and business value delivery.

Intended audience

This guide is intended for the following roles:

- Cloud architects planning the target Amazon EKS environment
- Platform engineers responsible for Kubernetes cluster operations
- DevOps engineers managing CI/CD pipelines and deployment workflows
- Migration leads coordinating the transition from self-managed clusters

Readers should have foundational knowledge of Kubernetes concepts (pods, deployments, services, namespaces), experience with kubectl, and familiarity with AWS core services.

Objectives

After reading this guide, you will be able to:

- Assess your current self-managed Kubernetes environment for migration readiness
- Select the appropriate migration approach (blue-green, phased, or lift-and-shift) based on your requirements
- Plan infrastructure, networking, storage, and security for the target Amazon EKS environment
- Define a testing and validation strategy that minimizes risk during cutover
- Identify post-migration optimization opportunities using AWS native services
- Communicate migration plans and progress effectively to stakeholders

Service availability

Some AWS services aren't available in all AWS Regions. For Region availability, see AWS services by Region. For specific endpoints, see the Service endpoints and quotas page, and choose the link for the service.

Prerequisites and limitations

Prerequisites

Requirement	Details
Python	3.9 or higher installed on the local machine
kubectl	Installed and configured with access to both the source cluster (for extraction) and the target Amazon EKS cluster (for migration)
AWS Command Line Interface (AWS CLI)	Version 2.x, configured with AWS Identity and Access Management (IAM) permissions for Amazon EKS, IAM, and Amazon EC2.
Helm (optional)	Version 3.x, required only if migrating Helm releases
Target Amazon EKS cluster	Must already exist. The migration script does not provision the cluster. For more informati

	on, see Creating an Amazon EKS cluster in the Amazon EKS documentation.
Amazon Elastic Block Store (Amazon EBS) CSI driver	Installed on the target Amazon EKS cluster for StorageClass provisioner remapping (`ebs.csi.aws.com`)
AWS Load Balancer Controller	Installed on the target Amazon EKS cluster for Service type LoadBalancer and Application Load Balancer Ingress support
CRD operators	Any controllers for custom resources used in the source cluster (for example, cert-manager, Prometheus Operator) must be installed on the target cluster before migration
IAM roles for service accounts (IRSA)	Configured on the target Amazon EKS cluster if workloads require AWS API access
Network connectivity	The local machine must have network access to both the source and target Kubernetes API servers
Cluster admin access	Admin-level access to both source and target clusters through kubectl. The extraction script requires read access to all resources; the migration script requires create/update permissions.

Limitations

- Persistent volume data is not migrated. Use AWS DataSync or Velero for persistent volume data migration.
- Helm releases require the original chart source to reinstall. The script outputs the commands but cannot run them without chart references.

- Secret values are created as opaque placeholders. You must populate actual values post-migration through AWS Secrets Manager, AWS Systems Manager Parameter Store, or External Secrets Operator.
- CRD operators and controllers must be pre-installed on the target Amazon EKS cluster before you migrate custom resource instances. For more information, see AWS Controllers for Kubernetes.
- If the source cluster uses a private container registry, ensure that Amazon EKS nodes have pull access, such as Amazon Elastic Container Registry (Amazon ECR) and image pull secrets.

Business challenges

Organizations face the following critical business challenges with self-managed Kubernetes.

Rising infrastructure management costs

Organizations spend significant budget on specialized Kubernetes engineers to keep clusters running. Teams dedicate substantial time to routine maintenance tasks like upgrades, patching, and troubleshooting control plane issues instead of building features that drive revenue. Migrating to Amazon EKS shifts the infrastructure management burden to AWS, reducing these operational costs.

Scalability bottlenecks

Self-managed clusters often struggle during rapid growth periods. Adding capacity requires hardware procurement, installation, and configuration that takes weeks or months. This delay prevents businesses from responding quickly to market opportunities or seasonal demand spikes. Amazon EKS enables faster scaling to meet business needs through managed node groups and cluster autoscaling.

Security and compliance pressure

Maintaining security patches, implementing compliance controls, and passing audits consumes enormous resources. A single missed security update can expose the entire organization to breaches. Managed security updates and AWS compliance certifications available through Amazon EKS reduce risk and audit burden.

Talent shortage and retention issues

Finding and keeping engineers with deep Kubernetes expertise is expensive and difficult. When key team members leave, they take critical knowledge with them, creating operational risks. Migrating to Amazon EKS reduces dependency on rare specialized skills by leveraging a managed service for control plane operations.

Limited integration with cloud services

Self-managed clusters require custom solutions to integrate with cloud databases, storage, monitoring, and identity systems. These integrations are fragile and require ongoing maintenance. Native AWS service integration in Amazon EKS eliminates these integration challenges.

Downtime and reliability concerns

Cluster failures can halt business operations, costing thousands per minute in lost revenue and damaged reputation. Self-managed setups require complex high-availability configurations that are difficult to maintain. Built-in reliability features of Amazon EKS, including a multi-AZ control plane, reduce downtime risk.

Slow innovation pace

When engineering teams spend time maintaining infrastructure, they have less capacity to focus on innovation that drives business differentiation. Migrating to Amazon EKS frees up technical resources to work on strategic initiatives rather than operational tasks.

Objectives

These are key objectives organizations should aim for when migrating to Amazon EKS:

- **Reduced operational overhead** - Decrease infrastructure management time significantly through automated cluster operations, managed control plane updates, and streamlined maintenance workflows.
- **Improved system reliability and uptime** - Achieve 99.95% or higher availability for the control plane with Amazon EKS SLA-backed reliability and automated health monitoring.
- **Faster time to market** - Accelerate deployment pipelines through automated CI/CD integration, streamlined workflows, and reduced manual intervention requirements.
- **Enhanced security posture** - Strengthen security through automated security patch management, continuous compliance monitoring, and integrated AWS security services.
- **Predictable and optimized costs** - Reduce total cost of ownership through right-sized infrastructure, automated resource optimization, and elimination of undifferentiated heavy lifting.
- **Seamless scalability** - Scale from hundreds to thousands of nodes within minutes with automated cluster scaling and optimized control plane performance.
- **Simplified disaster recovery** - Improve disaster recovery capabilities with automated backup strategies, multi-Region deployment options, and streamlined recovery procedures.
- **Better developer experience** - Accelerate developer onboarding and productivity through standardized managed infrastructure, consistent tooling, and reduced operational complexity.
- **Improved observability and troubleshooting** - Enhance issue detection and resolution through integrated Amazon CloudWatch monitoring, comprehensive logging, and native AWS observability tools.
- **Native AWS service ecosystem** - Simplify integration complexity through built-in AWS service support, seamless AWS Identity and Access Management (IAM) integration, and unified management experience.

Migration strategy

Moving your self-managed Kubernetes cluster to Amazon EKS requires careful planning and execution. This section provides a practical approach to planning and executing the migration.

Assessment and planning

Start by thoroughly understanding your current environment. Document all workloads, applications, dependencies, and configurations running on your cluster. Identify which applications are stateful versus stateless, as this affects migration complexity.

Create an inventory of:

- Number of nodes and pods
- Storage requirements and persistent volumes
- Network configurations and ingress rules
- Security policies and RBAC settings
- Custom controllers or operators
- Integration points with external systems

Choose your migration approach

For large clusters, you have several options:

- **Blue-green migration** — Build a complete new Amazon EKS cluster alongside your existing one. This approach minimizes downtime and allows easy rollback, making it ideal for production workloads. For more information, see Blue/Green Deployments on AWS.
- **Phased migration** — Move applications gradually in waves, starting with non-critical workloads. This reduces risk and allows you to learn from each phase. For more information, see Phases of a large migration with AWS.
- **Lift-and-shift** — Recreate your exact configuration in Amazon EKS, then migrate workloads. This works well when you want to maintain current architecture initially. For more information, see Migration strategies in AWS Prescriptive Guidance.

Infrastructure preparation

Set up your AWS foundation:

- Design your Amazon Virtual Private Cloud (Amazon VPC) with public and private subnets across multiple Availability Zones.
- Configure networking components (Elastic Load Balancing, Amazon Route 53, security groups).
- Set up IAM roles and policies for cluster access.
- Plan your storage strategy using Amazon Elastic Block Store (Amazon EBS), Amazon Elastic File System (Amazon EFS), or Amazon FSx.
- Establish monitoring and logging with CloudWatch.
- Use infrastructure as code, such as AWS CloudFormation, AWS Cloud Development Kit (AWS CDK), or Terraform, to provision your Amazon EKS cluster. This ensures consistency and repeatability.

Application migration process

For each application or workload group:

1. **Prepare** — Update container images if needed, adjust configurations for AWS services, and test compatibility. In most cases, containerized applications run on Amazon EKS without code changes. However, you may need to adjust Kubernetes manifests for Amazon EKS-specific features such as IAM roles for service accounts or AWS Load Balancer Controller annotations.
2. **Deploy to Amazon EKS** — Use your existing Kubernetes manifests with necessary modifications for Amazon EKS-specific features.
3. **Validate** — Run thorough testing to ensure functionality, performance, and integration points work correctly.
4. **Cut over** — Update DNS or load balancer configurations to route traffic to the new cluster.
5. **Monitor** — Watch metrics closely during and after migration to catch any issues early.

Data migration strategy

For stateful applications, plan data migration carefully. The migration scripts handle resource configurations (StorageClasses, PersistentVolumeClaims) but not the actual data stored in volumes. Persistent volume data requires separate migration using dedicated tools.

- Use AWS DataSync or Velero for persistent volume data migration.
- Consider database replication for zero-downtime migrations.
- Test data integrity after migration.
- Keep old data accessible during the transition period.

Testing and validation

Before going live:

1. Run integration tests in the new environment.
2. Perform load testing to verify performance.
3. Test disaster recovery procedures.
4. Validate security controls and compliance requirements.
5. Conduct user acceptance testing with key stakeholders.

Cutover and rollback planning

Plan your cutover window during low-traffic periods. Have a detailed rollback plan ready in case issues arise. For large clusters, consider:

- Gradual traffic shifting using weighted routing.
- Keeping the old cluster running for a defined period.
- Clear success criteria before decommissioning old infrastructure.

Post-migration optimization

Once migrated, take advantage of Amazon EKS features:

- Implement auto-scaling for nodes and pods.
- Use AWS-native services for logging and monitoring.
- Apply security best practices and compliance controls.
- Optimize costs by using Spot Instances and right-sizing.
- Set up automated backup and disaster recovery.

Best practices

- Run a dry-run migration before any live migration to catch misconfigurations, missing CRDs, and provisioner mismatches before they become problems on the target cluster.
- Review the extraction summary (SUMMARY.json) to verify resource counts and identify any unexpected resources before proceeding with migration.
- Use a phased migration approach for large clusters. Start with non-critical workloads, validate functionality, and then gradually move remaining applications.
- Capture baseline performance metrics before migration to enable accurate post-migration comparison.
- Follow the principle of least privilege and grant the minimum permissions required to perform a task. For more information, see Grant least privilege and Security best practices in the IAM documentation.
- Encrypt data in transit by using TLS for all Kubernetes API server communications and inter-service traffic.
- Install required CRD operators on the target Amazon EKS cluster before migrating workloads that depend on them.
- Keep the source cluster operational during and after migration to provide a rollback path until you have fully validated the new environment.
- Train your team on Amazon EKS-specific features and AWS integrations. Update runbooks and operational procedures before cutover.
- Monitor spending closely during migration. Use AWS Cost Explorer to track expenses and optimize resource usage.
- For clusters with hundreds of nodes and thousands of pods, engage AWS Support for guidance on planning, testing, and troubleshooting complex scenarios.
- Schedule the migration window during a low-activity period to minimize impact on dependent applications.

Key considerations for large-scale migrations

Minimize downtime

For clusters with hundreds of nodes and thousands of pods, plan carefully and test extensively. Use blue-green or phased migration approaches to maintain service availability throughout the transition.

Security

Leverage the AWS shared responsibility model. Amazon EKS manages control plane security while you handle workload security. Use IAM roles for service accounts (IRSA) to provide fine-grained AWS permissions to pods.

Scalability

The Amazon EKS control plane automatically scales with your workload demands, removing the operational burden of control plane capacity planning.

Team preparation

Train your team on Amazon EKS-specific features and AWS integrations. Update runbooks and operational procedures. Ensure on-call staff are familiar with Amazon EKS troubleshooting workflows.

Cost management

Monitor spending closely during migration. Use AWS Cost Explorer to track expenses and optimize resource usage. Consider Spot Instances for non-critical workloads and use Savings Plans for predictable baseline capacity.

Pre-migration checklist

This checklist outlines the steps required to migrate workloads from an existing Kubernetes cluster to Amazon EKS. It is organized into phases covering environment setup, source cluster assessment, target cluster preparation, networking, storage, security, monitoring, testing, migration planning, and post-migration activities. Each phase includes specific tasks that should be completed and verified before moving to the next. The checklist also includes rollback procedures and emergency contacts in case issues arise during the migration.

Phase 1: Environment setup

Software installation:

- Install Python 3.9 or higher.
- Install and configure kubectl.
- Install and configure the AWS CLI.
- Install Helm (if you are migrating Helm releases).
- Verify that all tools are accessible from the command line.

Access configuration:

- Configure kubectl access to the source cluster.
- Configure kubectl access to the target Amazon EKS cluster.
- Test switching between cluster contexts.
- Verify that AWS CLI credentials are configured.
- Confirm that IAM permissions for Amazon EKS, IAM, and Amazon EC2 are in place.

Phase 2: Source cluster assessment

Cluster inventory:

- Run the extraction script on the source cluster.
- Review the `SUMMARY.json` file for the cluster overview.
- Document the total number of namespaces.

- Count deployments, StatefulSets, and DaemonSets.
- Identify all custom resource definitions (CRDs).
- List all Helm releases.

Resource analysis:

- Document node specifications and capacity.
- Identify storage classes and persistent volumes.
- Review network policies and ingress configurations.
- List all secrets and ConfigMaps.
- Document RBAC roles and bindings.
- Identify external dependencies.

Application dependencies:

- List private container registries used.
- Document external database connections.
- Identify third-party service integrations.
- Note any custom operators or controllers.
- Review application-specific requirements.

Phase 3: Amazon EKS target cluster preparation

Cluster creation:

- Create the Amazon EKS cluster in the target AWS account.
- Configure the VPC with public and private subnets.
- Set up subnets across multiple Availability Zones.
- Configure security groups.
- Set up NAT gateways for private subnets.

Essential components installation:

- Install the Amazon EBS CSI driver.
- Install the AWS Load Balancer Controller.
  - Configure IAM roles for service accounts (IRSA).
- Install cert-manager (if used).
- Install Prometheus Operator (if used).
- Install any other custom CRD operators.

Node group configuration:

- Create managed node groups.
- Configure auto-scaling policies.
- Set appropriate instance types.
- Configure node labels and taints.
- Verify that nodes are ready and healthy.

Phase 4: Networking and storage

Network setup:

- Configure VPC endpoints for AWS services.
- Set up Amazon Route 53 DNS records.
- Configure load balancer settings.
- Review security group rules.
- Test connectivity between subnets.

Storage configuration:

- Verify the Amazon EBS CSI driver is functional.
- Create required storage classes. The migration script automatically converts old storage provisioners to modern CSI drivers (for example, `kubernetes.io/aws-ebs` becomes `ebs.csi.aws.com`).
- Test dynamic volume provisioning.
- Plan persistent volume data migration.

- Set up a backup solution (Velero or AWS Backup).

Phase 5: Security and secrets

IAM and RBAC:

- Create IAM roles for service accounts.
- Configure pod security policies.
- Review and update RBAC policies.
- Set up cluster authentication.
- Configure API server access controls.

Secrets management:

- Set up AWS Secrets Manager.
- Configure AWS Systems Manager Parameter Store.
- Plan the secret migration strategy.
- Document which secrets need manual population.
- Test secret retrieval from applications.

Phase 6: Monitoring and logging

Observability setup:

- Enable CloudWatch Container Insights.
- Configure CloudWatch Logs
- Set up log aggregation (Fluent Bit or Fluentd).
- Create CloudWatch dashboards.
- Configure alerting rules.
- Test monitoring data collection.
- Set up monitoring for both old and new clusters during the transition period to compare behavior.

Phase 7: Testing environment

Staging validation:

- Set up a staging Amazon EKS cluster.
- Run a dry-run migration on staging.
- Test sample workload deployment.
- Verify networking functionality.
- Test storage provisioning.
- Validate monitoring and logging.

Application testing:

- Deploy test applications.
- Run integration tests.
- Perform load testing.
- Test disaster recovery procedures.
- Validate security controls.

Phase 8: Migration planning

Documentation:

- Create a detailed migration runbook.
- Document rollback procedures.
- Prepare the communication plan for stakeholders.
- Schedule the migration window.
- Identify the on-call team members.

Data migration strategy:

- Plan persistent volume data migration.
- Set up DataSync (if needed).

- Configure Velero for backups.
- Test the data migration process.
- Verify data integrity checks.

Cutover planning:

- Define success criteria.
- Create a cutover checklist.
- Plan the DNS switching strategy.
- Schedule traffic routing changes.
- Prepare rollback triggers.

Phase 9: Final validation

Pre-migration checks:

- Verify that all prerequisites are met.
- Confirm that the target cluster is ready.
- Test migration scripts in staging.
- Review extraction output completeness.
- Validate team readiness.

Communication:

- Notify stakeholders of the migration schedule.
- Brief the operations team on the procedures.
- Prepare status update templates.
- Set up communication channels.
- Document escalation paths.

Phase 10: Prepare for post-migration needs

Validation plan:

- Prepare application health checks.
- Create validation test scripts.
- Document expected metrics.
- Plan user acceptance testing.
- Prepare performance benchmarks.

Cleanup strategy:

- Plan the old cluster decommissioning timeline.
- Document the resource cleanup procedures.
- Schedule a cost review.
- Plan optimization activities.
- Archive the migration documentation.

Emergency procedures

Rollback readiness:

- Document rollback decision criteria.
- Test rollback procedures.
- Prepare DNS rollback commands.
- Keep the source cluster operational.
- Maintain backup access paths.

Support contacts:

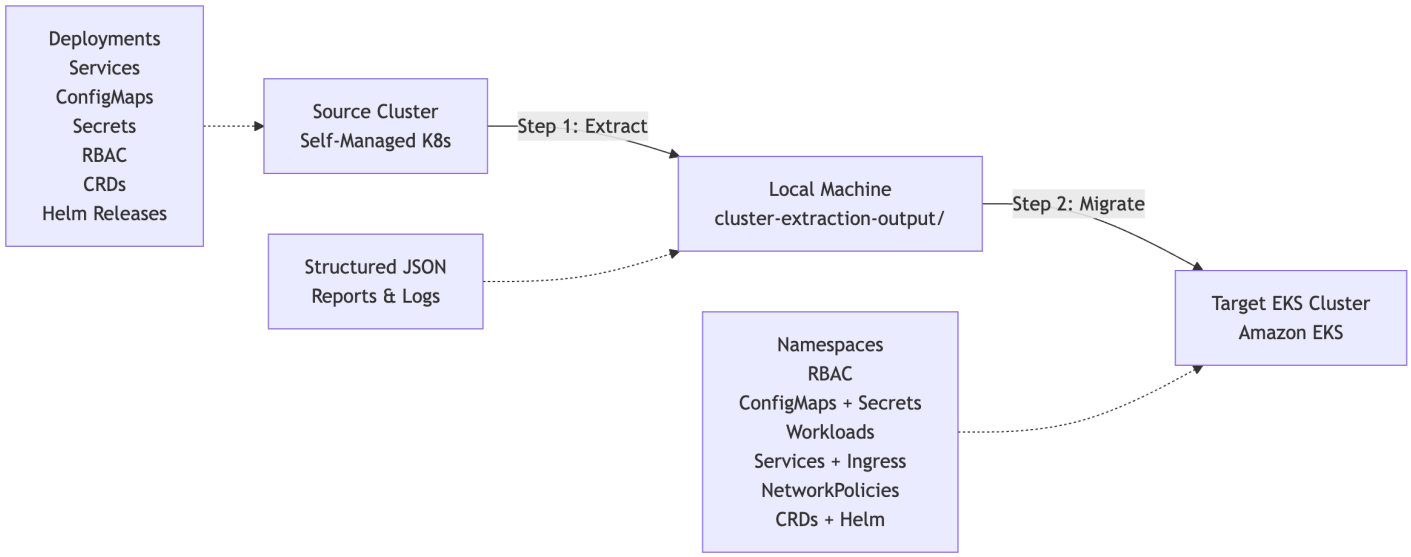
- AWS Support case number (if engaged).
- Internal escalation contacts.
- Vendor support contacts.
- Team member contact list.
- Emergency communication plan.

Migration procedures

Moving your workloads from a self-managed Kubernetes cluster to Amazon EKS happens in two main steps: extraction and migration. This approach gives you full visibility into what is being moved and lets you validate everything before making changes.

For the code repository that implements this migration, see [sample-self-Managed-Kubernetes-Cluster-to-Amazon-EKS-migration] on GitHub.

How the solution works



The migration flow consists of two stages:

1. **Extract** — Pull all resources from the source cluster into structured JSON.
2. **Migrate** — Transform resources for Amazon EKS compatibility and deploy them to the target cluster with validation.

Step 1: Extract everything from your current cluster

The extraction script pulls a complete snapshot of your existing cluster. It walks through your entire environment systematically, capturing ten categories of resources: cluster information, namespaces, workloads, networking, storage, ConfigMaps, RBAC, CRDs, Helm releases, and resource policies. Everything is saved as structured JSON files in organized directories.

The script captures secret metadata only—the names and types of secrets, not the actual values. This keeps sensitive data secure during extraction.

Procedure:

1. Switch kubectl context to the source cluster:

```
kubectl config use-context <source-cluster-context>
```

2. Clone the repository and run the extraction script:

```
git clone https://github.com/aws-samples/sample-self-Managed-Kubernetes-Cluster-to-Amazon-EKS-migration
cd extraction/
python extract_cluster_info.py
```

3. Review the extraction summary. All output is written to `cluster-extraction-output/` as structured JSON.

What the extraction captures:

Step	Resources	Output path
1	Kubernetes version, node info (capacity, labels, taints, runtime)	`cluster/`
2	Namespaces (excludes kube-system, kube-public, kube-node-lease)	`cluster/`
3	Deployments, StatefulSets, DaemonSets, Jobs, CronJobs	`workloads/{ns}/`
4	Services, Ingresses, NetworkPolicies	`networking/{ns}/`
5	StorageClasses, PersistentVolumes, PVCs	`storage/`

6	ConfigMap data (full key-value pairs), Secret metadata (no values)	`config/{ns}/`
7	ClusterRoles, ClusterRoleBindings, Roles, RoleBindings, ServiceAccounts	`rbac/`
8	Custom Resource Definitions and their instances	`crds/`
9	Helm release metadata (name, namespace, version, status)	`helm/`
10	ResourceQuotas, LimitRanges	`policies/{ns}/`

Step 2: Migrate to your Amazon EKS cluster

The migration script takes the extracted JSON output and deploys everything to your target Amazon EKS cluster in eight sequential phases—namespaces first, then RBAC, storage, config, CRDs, workloads, networking, and finally Helm releases. Each phase validates the result before moving to the next one. The script supports a dry-run mode so you can preview exactly what will happen before applying any changes, and it can resume from a specific phase if something needs fixing mid-migration.

The script automatically filters out system resources that conflict with Amazon EKS management, including default namespaces, system cluster roles, and service account token secrets. The migration also automatically adjusts networking configurations—it removes old cluster IPs, adds AWS Load Balancer Controller annotations, and configures ingresses for Application Load Balancers.

Procedure:

1. Switch kubectl context to the target Amazon EKS cluster:

```
kubectl config use-context <eks-cluster-context>
```

Or use the AWS CLI:

```
aws eks update-kubeconfig --name <cluster-name> --region <region>
```

2. Verify with:

```
kubectl get nodes
```

3. Run a dry-run migration:

```
cd migration/
python migrate_to_eks.py --dry-run
```

4. Review the dry-run report saved to `migration-output/dry-run/migration_report.json`. Check for errors, skipped resources, and warnings. Resolve any issues before proceeding with the live migration.
5. Run the full migration:

```
python migrate_to_eks.py --extraction-dir ../extraction/cluster-extraction-output
```

6. Resume migration from a specific phase (if needed):

```
python migrate_to_eks.py --start-phase 4 --extraction-dir ../extraction/cluster-
extraction-output
```

Validation

Throughout the migration, detailed logs capture everything that happens. When the migration finishes, you get a comprehensive JSON report showing what migrated successfully, what was skipped, and any errors encountered.

Sample migration report:

```
{
  "phases": [
    {
      "phase": "1-namespaces",
      "status": "completed",
```

```
      "migrated_count": 12,
      "skipped_count": 4,
      "error_count": 0,
      "migrated": ["app-prod", "app-staging", "..."],
      "skipped": ["default", "kube-system", "..."],
      "errors": []
    }
  ],
  "dry_run": false
}
```

Post-migration validation tasks:

Task	Description
Review the migration report	Open `migration-output/live/migration_report.json` . Verify that all phases show `"status": "completed"` with expected `migrated_count` values and zero or acceptable `error_count` .
Verify namespaces	Run `kubectl get namespaces` on the target Amazon EKS cluster. Confirm that all application namespaces from the source cluster are present.
Verify RBAC resources	Run `kubectl get clusterroles,clusterrolebindings,roles,rolebindings,serviceaccounts --all-namespaces` and spot-check that migrated RBAC resources exist and have the correct bindings.
Verify storage classes	Run `kubectl get storageclasses` . Confirm that migrated StorageClasses use the correct Amazon EKS-compatible provisioners (for example, `ebs.csi.aws.com`).
Verify ConfigMaps and secrets	Run `kubectl get configmaps,secrets --all-namespaces` . Confirm that

	ConfigMaps contain the expected data. Verify that placeholder secrets exist and note which ones need real values populated.
Verify workload health	Run `kubectl get deployments,statefulsets,daemonsets --all-namespaces` . Confirm that all workloads are in ready state. For Deployments, verify that the desired and available replica counts match. Run `kubectl get pods --all-namespaces` to check for CrashLoopBackOff or ImagePullBackOff errors.
Verify services and ingress	Run `kubectl get services,ingresses --all-namespaces` . For LoadBalancer services, confirm that an external IP or hostname has been assigned by the AWS Load Balancer Controller. For Ingresses, verify that the Application Load Balancer has been provisioned.
Verify CRDs	Run `kubectl get crds` and spot-check custom resource instances. Confirm that CRD controllers are reconciling the resources correctly.
Test application connectivity	Perform end-to-end application testing. Verify that services are reachable, inter-service communication works, and external endpoints respond correctly.
Populate secret values	Populate the placeholder secrets with actual values using Secrets Manager, AWS Systems Manager Parameter Store, or External Secrets Operator. Verify that workloads can access the secrets and function correctly.

Reinstall Helm releases	Use the `helm install` commands generated in Phase 8 of the migration to reinstall Helm releases from their original chart sources. Verify that Helm-managed resources are healthy.
Update DNS and traffic routing	Update DNS records, external load balancers , or traffic routing (for example, Route 53 weighted routing) to point to the new Amazon EKS cluster endpoints. Consider a gradual traffic shift for critical workloads.

Rollback and cleanup

If issues arise after migration, use the rollback script to remove migrated resources from the target Amazon EKS cluster:

1. Switch kubectl context to the target Amazon EKS cluster:

```
aws eks update-kubeconfig --name <cluster-name> --region <region>
```

2. Run a dry-run rollback:

```
cd rollback/
python3 rollback_eks_migration.py --dry-run
```

3. Review the dry-run rollback output. Confirm that only the expected migrated resources are listed for deletion. The rollback proceeds in reverse order: networking → workloads → CRDs → config → storage → RBAC → namespaces.
4. Run the full rollback:

```
python3 rollback_eks_migration.py
```

5. Review the rollback report. The `rollback_report.json` file is saved alongside the migration report with details of what was deleted, what was skipped, and any errors encountered.
6. Verify cleanup by running these commands to confirm that all migrated resources have been removed from the target Amazon EKS cluster:

```
kubectl get namespaces
kubectl get deployments --all-namespaces
kubectl get services --all-namespaces
```

Troubleshooting Kubernetes workload migrations to Amazon EKS

Issue	Resolution
kubectl cannot connect to the source cluster	Verify the kubeconfig context is correct: `kubectl config current-context` . Check network connectivity and authentic ation credentials.
Extraction script fails with permission errors	Ensure the kubectl service account has cluster-admin or equivalent read permissions on the source cluster.
Migration dry-run shows CRD errors	Install the required CRD operators on the target Amazon EKS cluster before running the migration. The migration script does not install operators.
Workloads stuck in Pending state after migration	Check node capacity on the target Amazon EKS cluster (`kubectl describe nodes`). Verify that resource requests can be satisfied. Check for missing StorageClasses or PVCs.
Services not getting external IPs	Verify that the AWS Load Balancer Controller is installed and running. Check IAM permissio ns for the controller's service account.
ImagePullBackOff errors on migrated pods	Ensure Amazon EKS nodes have access to the container registry used by the source cluster. Configure Amazon ECR pull-through cache or image pull secrets as needed.
StorageClass provisioner mismatch	The migration script automatically remaps `kubernetes.io/aws-ebs` to `ebs.csi.a ws.com` . For other provisioners, manually

	update the StorageClass before migration or after extraction.
Rollback script cannot find migration report	Specify the report path explicitly: `python3 rollback_eks_migration.py --report <path-to-migration_report.json>` .

Next steps

- **Review Amazon EKS best practices -** See the Amazon EKS Best Practices Guide for operational, security, and cost optimization guidance.
- **Engage AWS Support** - For large clusters with hundreds of nodes and thousands of pods, open a support case for migration planning assistance.
- **Explore training -** Consider AWS Training and Certification courses on Amazon EKS and container orchestration to prepare your team.
- **Plan ongoing optimization -** After migration, schedule regular reviews of cost, performance, and security posture by using AWS-native tools.
- **Modernize incrementally:** Use the migration as an opportunity to adopt cloud-native practices, such as IAM roles for service accounts, Amazon CloudWatch Application Insights, and managed node groups.

Resources

References

- Amazon EKS User Guide
- Amazon EKS Best Practices Guide
- Amazon EBS Driver
- AWS Load Balancer Controller
- IAM roles for service accounts
- Creating an Amazon EKS cluster

Tools

- sample-self-Managed-Kubernetes-Cluster-to-Amazon-EKS-migration (GitHub)
- AWS DataSync
- Velero

Guides and patterns

- Migration strategies (AWS Prescriptive Guidance)
- Blue/Green Deployments on AWS
- Phases of a large migration with AWS

Document history

The following table describes significant changes to this guide.

Change	Description	Date
Initial publication	—	June 8, 2026