SUSECON 25

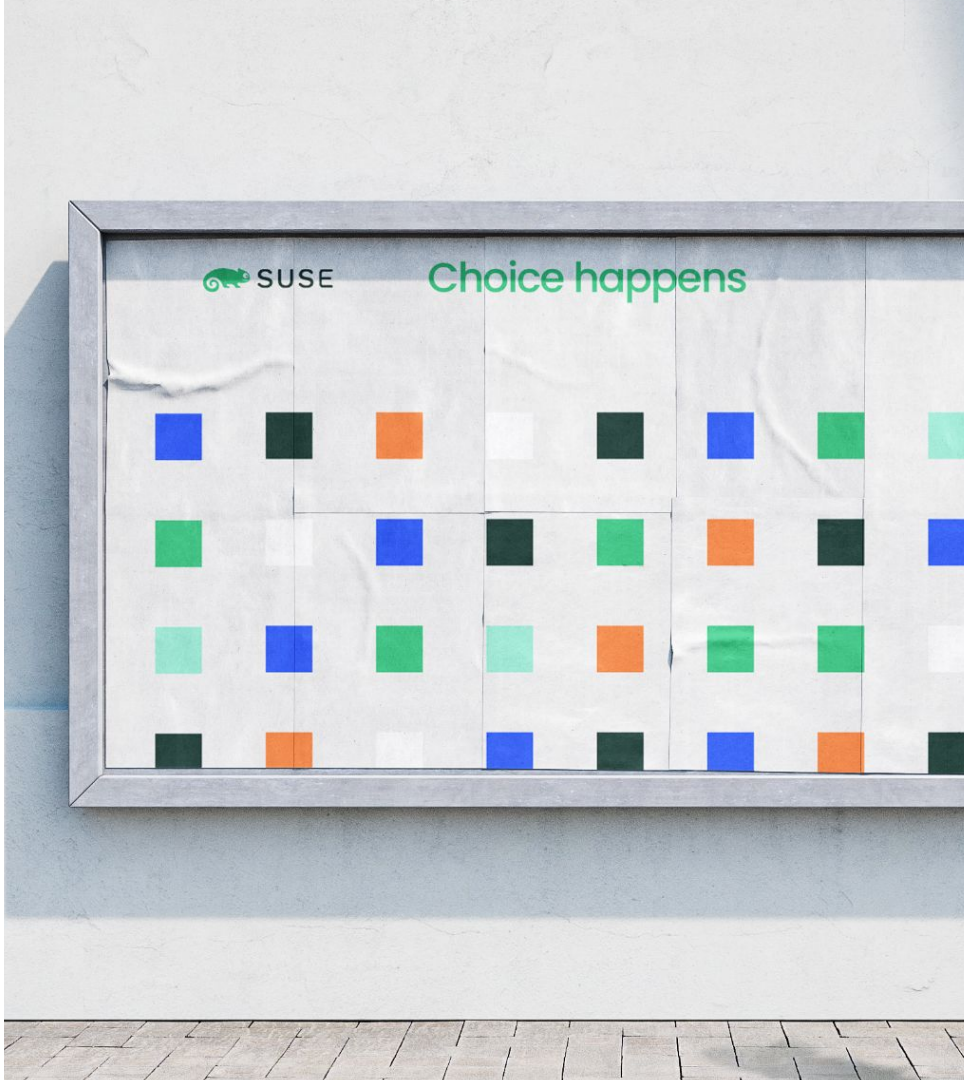
Migrating from RKE1 to RKE2

TUTORIAL-1096

Matthew Mattox

Agenda

- Why Migrate from RKE1 to RKE2?
- Managing Multi-Cluster Environments with **Rancher**
- Migration Strategies: **Lift-and-Shift | Rolling | Phased**
- Persistent Data Migration with **Longhorn & pv-migrate**
- **Demo:** Workload Migration from RKE1 → RKE2
- Troubleshooting Migration Challenges
- **Q&A & Discussion**



Matthew Mattox

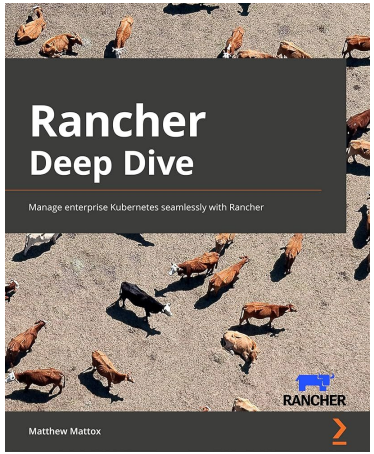
Principal Support Engineer at SUSE (6Yrs)

Training:rancher.academy

GitHub: github.com/supporttools

Blog: support.tools

Author of Rancher Deep Dive



Kubestronaut

Why Migrate from RKE1 to RKE2?

RKE1 End of Life (EOL) - July 31, 2025
Official SUSE Announcement: https://www.suse.com/support/kb/doc/?id=000021513

- **RKE1 will no longer receive updates, security patches, or support after July 31, 2025.**
- Organizations must transition to **RKE2** to maintain a supported and secure Kubernetes environment.
- **Delaying migration increases operational risk** due to lack of security updates and compatibility issues with future Kubernetes versions.

Why RKE2?

- Improved Security: SELinux support, FIPS compliance
- Better Performance: Containerd, optimized networking
- Long-Term Stability: Upstream Kubernetes alignment
- Rancher Integration: Multi-cluster management, rolling upgrades

More Reasons You Might Need to Migrate Your Kubernetes Clusters

- Moving into and out of the cloud
  Migrate workloads between cloud providers or transition on-prem to cloud for cost savings, compliance, or flexibility.
- Disaster Recovery (DR) & High Availability
  Ensure business continuity by maintaining a failover cluster or running workloads across multiple regions.
- Foundational Changes & Infrastructure Upgrades
  Upgrade Kubernetes versions, storage backends, or networking stacks with minimal downtime.

Moving Into and Out of the Cloud

Organizations move workloads between cloud and on-prem for cost, compliance, performance, and vendor flexibility.

Challenges:

- **Networking Differences** – VPC configurations, CNI plugins, and ingress controllers may need reconfiguration.
- **Cloud Storage Differences** – Persistent volume formats are cloud-specific (e.g., AWS EBS vs. Azure Disks).
- **IAM & Security Policies** – Role-based access control (RBAC) and firewall rules need to be updated.

Example Use Cases:

- **AWS EKS → RKE2 on-prem** for cost control and compliance.
- **Migrating from self-managed Kubernetes → managed services** (EKS, AKS, GKE).
- **Hybrid & Multi-Cloud Scaling** – Distribute workloads across cloud and on-prem for resilience.

Disaster Recovery (DR) & High Availability

Why It Matters

- **Minimize downtime** – Ensure workloads remain available during failures.
- **Protect against outages** – Cloud, network, or hardware failures can impact critical services.
- **Regulatory compliance** – Many industries require **business continuity** plans.

Challenges

- **Keeping Stateful Applications in Sync** – Databases and persistent volumes need consistent replication.
- **Failover Orchestration** – Switching traffic between clusters using DNS, load balancers, or BGP.
- **Storage & Data Replication** – Ensuring persistent data (PVs, databases) is **available in both clusters**.

Example Use Cases

- **Active-Passive Setup** – A secondary cluster on standby, ready to take over.
- **Geo-Redundant Deployments** – Workloads run across multiple regions for fault tolerance.
- **Automated DR Testing** – Regular failover simulations to validate disaster recovery plans.

Foundational Changes & Infrastructure Upgrades

Why Migrate?

- **Adopt New Kubernetes Architectures** – Move from outdated environments to modern, secure, and optimized clusters.
- **Improve Performance & Scalability** – Upgrade networking, storage, and runtime components for efficiency.
- **Enhance Security & Compliance** – Implement **Pod Security Standards (PSS)**, **SELinux**, and **RBAC improvements**.

Challenges

- **Migrating Workloads While Maintaining Dependencies** – Ensuring ConfigMaps, Secrets, and RBAC roles are properly transferred.
- **Networking & Ingress Controller Changes** – Switching from **Flannel** → **Cilium** or upgrading **Ingress-NGINX** can impact services.
- **Storage Backend Migration** – Transitioning from **NFS** → **Longhorn** or **Ceph** → **Rook** without data loss.

Example Use Cases

- **Switching Container Runtimes** – Migrating from **Docker** → **Containerd** for Kubernetes compatibility.
- **Upgrading Storage Solutions** – Moving from traditional NFS storage to cloud-native **Longhorn** for better resilience.
- **Kubernetes API & Version Changes** – Ensuring compatibility with deprecations and new security policies.

Choosing the Right Migration Strategy

Why Migration Strategy Matters

- Migrating workloads between clusters isn't a **one-size-fits-all** process.
- The right strategy depends on:
  - **Timeline** – How fast do you need to move?
  - **Risk Tolerance** – Can you afford downtime or need a gradual transition?
  - **Team Involvement** – Will this be **admin-driven** or do **app teams need control**?
  - **Cluster Differences** – Are you making minimal changes or a major infrastructure shift?

Three Common Strategies for Cluster Migration

- **Lift-and-Shift** – Move everything **at once**, minimal changes.
- **Rolling Migration** – Move **one application at a time**, working with app teams.
- **Phased Migration** – **Build a new cluster**, let app teams migrate at their own pace.

Each approach has trade-offs between speed, risk, and control. Let's break them down.

Lift-and-Shift Migration

What is Lift-and-Shift?

- **You**, as the **cluster admin**, move **all workloads** from one cluster to another **in one big move**.
- **Little to no changes** are made to applications or configurations.
- Works best when **workloads are compatible** with the new cluster.

Pros

- **Fastest migration method** – Everything moves at once.
- **Minimal involvement from app teams** – Admin-driven.
- **Works well when clusters are nearly identical** (same Kubernetes version, storage, etc.).

Cons

- **Higher risk of failures** – No gradual testing phase.
- **Potential downtime** – Some workloads may need to restart in the new cluster.
- **Infrastructure differences** – Network, storage, and security differences may require post-move fixes

Rolling Migration

What is Rolling Migration?

- **You**, as the **cluster admin**, move **applications one at a time** in coordination with the **app teams**.
- You, make small to medium size changes to the applications in-order to better use the new environment.
- Each app team **tests, updates, and validates** their services in the new cluster before fully migrating.
- **More controlled approach**, reducing risk and ensuring stability.

Pros

- **Minimized risk** – Applications are moved gradually.
- **App teams validate their own workloads** – Less troubleshooting after migration.
- **No major downtime** – Old cluster stays online while workloads migrate.

Cons

- **Slower migration process** – Requires **coordination with multiple teams**.
- **Potential inconsistencies** – If teams don’t migrate in sync, dependencies may break.
- **Higher resource costs** – Both clusters **run in parallel** until the migration is complete.

Phased Migration

What is Phased Migration?

- **You**, as the **cluster admin**, **build a new cluster** and inform **app teams** that they need to migrate.
- The **responsibility is on app teams** to move their workloads **when ready**.
- **Cluster stays online** until everything is moved, then the old cluster is decommissioned.

Pros

- **Less work for cluster admins** – App teams handle their own migrations.
- **Flexibility** – Teams move **on their own timeline**, reducing coordination pressure.
- **Great for major infrastructure changes** – Teams can refactor if needed before moving.

Cons

- **Unpredictable timeline** – Some teams may **delay migration**, leaving **two clusters running longer**.
- **Potential inconsistencies** – If teams don't migrate in a structured way, **dependencies may break**.
- **May require temporary workarounds** – Cross-cluster communication might be needed during migration.

Migration Methods – Choosing the Right Approach

Why Choosing the Right Migration Method Matters

- Different workloads and environments require **different migration techniques**.
- Key factors to consider:
  - **Are your workloads stateless or stateful?**
  - **Do you need a fast migration or a controlled process?**
  - **How critical is data consistency?**

Common Migration Methods:

- **YAML Export/Import** – Quick and simple method for stateless workloads.
- **DR-Syncer** – A tool designed to **sync namespaces and Persistent Volumes across clusters**.
- **Velero / CloudCasa** – Best for full-cluster backups and restores.
- **Redeploy** – Using GitOps or CI/CD pipelines to provision workloads from scratch in a new cluster.

Redeploy

How It Works

- **Update the target cluster in your pipelines** to reflect the new environment.
- **Deploy a fresh environment** in the new cluster using Helm, Kustomize, or GitOps (ArgoCD, Flux).
- **Migrate data separately** using snapshots, database replication, or manual restores.

Pros

- **Ensures a clean deployment**, avoiding legacy config issues.
- **Best for infrastructure upgrades** or Kubernetes version changes.

Cons

- **No automatic PV migration**, must handle database and storage manually.
- **Takes more time**, especially for complex applications.
- **Requires applications to be fully defined as code (IaC/GitOps).**

Best for:

- **Organizations following Infrastructure-as-Code (IaC) or GitOps practices.**
- **Teams migrating to declarative deployments for better reproducibility.**

YAML Export/Import

How It Works

- **Export workloads** using:
  `kubectl get resource -o yaml > backup.yaml`
- **Apply them in the new cluster** with:
  `kubectl apply -f backup.yaml`

Pros

- **Fast and simple**, no extra tools required.
- **Good for stateless workloads** (Deployments, Services, ConfigMaps).

Cons

- **No Persistent Volume (PV) migration**, must move storage separately.
- **Manual and error-prone**, requires dependency handling.

Best for: Small workloads, **quick transitions**, and environments without persistent data.

Open Source Tool:

GitHub: mattmattox/kubebackup

Velero / CloudCasa

How It Works

- **Backup workloads** in the old cluster using Velero or CloudCasa.
- **Restore them** in the new cluster, including Persistent Volumes.

Pros

- **Works across cloud and on-prem clusters**.
- **Backs up all workloads**, including PVs, RBAC, and secrets.

Cons

- **Requires object storage** (AWS S3, MinIO, Azure Blob).
- **May be slow** for large clusters with many Persistent Volumes (PVs).
- **CloudCasa is a paid service and requires a license.**

Best for:

- Full-cluster migrations needing persistent storage and security settings.
- Backup and disaster recovery strategies.

https://cloudcasa.io/blog/seamless-migration-from-rke-to-rke2-with-cloudcasa-a-suse-partner-solution/

DR-Syncer

How It Works

- Replicates **Deployments, Services, ConfigMaps, Secrets, Persistent Volumes, etc.** across clusters.
- Ensures **scheduled syncing** for seamless migration.

Pros

- **Purpose-built for Kubernetes migrations/DR** – Handles both workloads and PVs.
- **Minimizes downtime** – Keeps namespaces and data **synchronized** across clusters.
- **More efficient than manual YAML exports** – Reduces human error.

Cons

- **Requires setup & configuration** – Not a native Kubernetes feature.
- **May need cluster connectivity** – Ensure network policies allow cross-cluster syncs.
- **Requires similar cluster setup** – The target cluster should match the source.
- **Target cluster must have storage configured** – Ensure PVs can be replicated properly.

Best for: Stateless and Stateful applications that need replication between clusters.

Open Source Tool: GitHub: supporttools/DR-Syncer

Data Migration with Longhorn

How It Works

- **Longhorn's Disaster Recovery (DR) volumes** sync with a backup cluster **on a scheduled basis** using **incremental restores**.
- The DR volume is created from **a volume's backup in the backupstore**.
- **Scheduled backup intervals** determine how frequently data is updated.
- If the original volume fails, **the DR volume can be activated in the backup cluster** for fast recovery.

Pros

- **Scheduled Data Syncing** – Uses periodic snapshots and incremental restoration.
- **Faster Recovery vs. Full Backup Restores** – Avoids the need to recover the entire volume from scratch.
- **Built-in with Longhorn** – No additional tools required for **Longhorn users**.

Cons

- **Not real-time replication** – Data is **only as current as the last scheduled backup**.
- **No live snapshots or backups on DR volumes** – Backups must be restored before activation.
- **Recovery Point Objective (RPO) depends on backup frequency** – If backups run hourly, up to **one hour of data could be lost** in case of failure.

Best for:

- **Organizations already using Longhorn** for persistent storage.
- **Disaster Recovery (DR) planning** where **scheduled snapshots are acceptable**.
- **Workloads that can tolerate some data loss (RPO > 0)** in exchange for lower Recovery Time Objective (RTO).

Data Migration with pv-migrate

How It Works

- **pv-migrate** is a **CLI tool** that copies **Persistent Volume Claims (PVCs)** across **namespaces, clusters, or storage backends**.
- Uses **rsync over SSH** with **Load Balancers, Bind Mounts, and Port-Forwarding** for data transfer.
- Supports **multiple migration strategies**, **automatically selecting** the most efficient method based on the environment.

Pros

- **Works across namespaces, clusters, and storage backends** – Not tied to a specific CSI driver.
- **Secure migrations** – Uses **SSH and rsync** for encrypted data transfer.
- **Multiple migration strategies** – Falls back to different approaches when needed.
- **Highly customizable** – Users can configure **rsync/SSH images, affinity, and network settings**.

Cons

- **Requires storage compatibility** – Target storage class **must support the expected access modes**.
- **Live data requires careful handling** – Works best for **pre-migration syncing**, not real-time replication.
- **Networking considerations** – **Cross-cluster migrations require proper network connectivity** between clusters.

Best for:

- **Moving Persistent Volumes across namespaces or clusters.**
- **Changing storage classes** (e.g., ReadWriteOnce → ReadWriteMany).
- **Migrating data securely between Kubernetes clusters.**
- **Expanding PVCs when volume resizing isn't supported.**

Open Source Tool: GitHub: utkuozdemir/pv-migrate

Cattle-Drive

How It Works

A tool to migrate Rancher objects created for downstream cluster from a source to a target cluster, these objects include, but not limited to:

- **Projects**
- **Namespaces**
- **Rancher Permissions**
- **Cluster Apps / Catalog Repos**

Pros

- **Automates the Rancher resources between clusters**

Cons

- **Does not migrate your applications**

Best for:

- **With redeployment migrations where you don’t want to manually recreate the Projects and permissions.**

https://github.com/rancherlabs/cattle-drive

Common Migration Failures & Fixes (Part 1)

Missing Critical Cluster Services

Issue: After migration, applications fail due to **missing dependencies** like **cert-manager, monitoring, or GitOps tools**.
Fix:

- **Ensure required cluster services are installed first** (cert-manager, Prometheus, ArgoCD).
- Deploy **cluster-wide services before migrating workloads**.

Forgetting Cluster-Scoped Resources (ClusterRoles, CRDs, etc.)

Issue: Applications fail to start because **ClusterRoles, RoleBindings, or CRDs are missing**.

Fix:

- **Export and apply CRDs before migrating workloads:**

  ```
  kubectl get crd -o yaml > crds.yaml

  kubectl apply -f crds.yaml
  ```

- Ensure **RBAC rules (ClusterRoleBindings, ClusterRoles) are migrated properly**.
  List cluster-wide resources with:

  ```
  kubectl api-resources --verbs=list --namespaced=false
  ```

Common Migration Failures & Fixes (Part 2)

Secrets Not Stored Externally

Issue: Applications crash because **Secrets were lost** during migration.

- **Externalize secrets** using Vault, AWS Secrets Manager, or Kubernetes External Secrets.
- **Backup secrets before migration:**
  `kubectl get secrets -A -o yaml > secrets-backup.yaml`
- **Restore secrets manually** or via GitOps after migration.

CNI Changes Impact Network Policies

Issue: A **different CNI (Calico, Cilium, etc.)** can change **network policies**, causing communication failures.

- **Check existing network policies** before migration:
  `kubectl get networkpolicy -A`
- **Verify pod-to-pod and pod-to-service communication is allowed.**
- **Update network policies to match the new CNI's behavior** before migration.

Best Practices to Avoid Issues

- **Pre-flight validation** – Run `kubectl get all -A` to identify missing resources.
- **Use GitOps (ArgoCD/Flux) to store and redeploy cluster-wide resources.**
- **Test migration in a staging cluster before production cutover.**
- **Document all external services and cluster-scoped dependencies before migration.**

DEMO

Q&A

Thank You!


SUSE
SUSECON 25