



REFERENCE ARCHITECTURE

CloudCasa Self-Hosted and Red Hat OpenShift Virtualization

Self-Hosted Version

Table of Contents

Table of Contents	2
Executive Overview	5
What This Document Covers	5
Scope and Assumptions.....	6
Red Hat OpenShift and Virtualization	6
VMware vs. Red Hat OpenShift Virtualization	6
vSphere to OpenShift Virtualization Concept Mapping	6
Architecture / Diagrams	7
Component Architecture	7
CloudCasa Self-Hosted Overview	7
CloudCasa Server Architecture.....	8
CloudCasa Agents on Protected Clusters.....	10
CloudCasa Agent Installation	10
Network Flow.....	13
Data Movement (File Index / CSI Snapshots).....	13
Storage Topology	15
Source PVC/PV (Block Ceph / File NFS)	15
Storage Prerequisites for VM Workloads.....	15
Storage Validation.....	16
Backup Storage Locations.....	16
Encryption	17
Object Lock and Immutable Backups.....	17
Functionality	17
VM Backup / Restore	17
VM Protection Scope (OCP-V)	20
Policy Examples.....	20
Replication and Migration.....	21
Replication	22
Migration	22

Migration Transforms	22
Granular File Recovery	23
Application Support	24
Disaster Recovery	25
Introduction.....	25
Inventory PVC	26
Failover	26
What happens during failover.....	26
Failback.....	28
Disaster Recovery of the CloudCasa Server.....	28
Adding a remote CloudCasa server	29
Automatic reverse configuration	30
Network connectivity requirements	30
Recommended Scenarios	31
Two Control Plane Topologies.....	31
Pattern A: Dedicated Management Cluster (Recommended for Production)	32
Pattern B: Co-Located on the Workload Cluster.....	34
Recommended Production Layout.....	35
Primary and Secondary Site Layout	36
Permutations and Other Patterns	36
CloudCasa Server Placement and Protection	36
Workload Cluster Topologies.....	37
DR Testing Considerations	38
Catalog Backup	38
Backup of the Catalog: The Most Critical Job	38
Disaster Recovery Strategy	38
RTO and RPO	39
DR Testing	40
Performance Tuning.....	40
Memory Utilization / Configuration Recommendations	40
Appendix: Additional Operational and Security Considerations.....	42
Authentication and Authorization.....	42

GitOps Lifecycle with Red Hat OpenShift GitOps (ArgoCD).....	43
Provisioning CloudCasa with Terraform	43
Hardening with Red Hat Advanced Cluster Security (RHACS)	44
Air-Gapped Installation	44
Network Configuration	45
Network.....	45
CloudCasa Configuration	45
Helm Values Overview.....	45
Site B Import / Restore.....	46
Day-2 Operations Checklist.....	46
Monitoring	47
References	48

Executive Overview

Red Hat OpenShift has become a default platform for enterprise cloud-native orchestration. With Red Hat OpenShift Virtualization, virtual machines run alongside containerized applications on the same KVM-based platform, which lets infrastructure teams consolidate VMware-era workloads onto the same control plane they use for cloud-native apps.

OpenShift provides day-1 and day-2 capabilities for the platform itself. It does not provide application-aware backup, cross-cluster mobility, or ransomware recovery for the workloads running on it. That is the gap CloudCasa fills.

Through its Software as a Service (SaaS) and Self-hosted model options versions, CloudCasa can provide flexibility for a user's data protection requirements. It can run entirely inside the customer's environment without any external dependencies. CloudCasa agents run on protected clusters and communicate only to a customer-owned CloudCasa server; the server stores its catalog in customer-owned MongoDB; and backup data lands in customer-owned S3-compatible, NFS, or SMB storage. This makes it the right fit for air-gapped, sovereign, or regulated environments where data residency and offline operation are non-negotiable.

Whether you're simply protecting Red Hat OpenShift / Red Hat OpenShift Virtualization, looking to integrate disaster recovery capabilities into your data protection strategies, or simply looking for better governance and restore capabilities your cloud native infrastructure, CloudCasa can be a great platform for simplifying these day-2 operations.

This document describes a reference architecture for protecting Red Hat OpenShift and OpenShift Virtualization workloads with CloudCasa Self-Hosted, deployed on-premises only, with both a primary and a secondary on-premises site.

What This Document Covers

- Component model for CloudCasa Self-Hosted and how it maps onto Red Hat OpenShift.
- Two deployment topologies: a dedicated management cluster, and management co-located with workloads.
- A primary plus secondary on-premises DR architecture with no public cloud dependencies.
- Backup, recovery, application and VM mobility, and ransomware protection use cases for OCP and OCP-V.
- Storage prerequisites for protecting OpenShift Virtualization VMs (CSI, RWX block, snapshot class).

- Operational configuration: authentication, Role Based Access Control (RBAC), catalog backup, ArgoCD lifecycle, Red Hat Advanced Cluster Security (RHACS) hardening.

Scope and Assumptions

This reference architecture assumes a self-hosted deployment with no public cloud sites. Variants that target ROSA, ARO, or hybrid configurations are out of scope for this document. The architecture assumes OpenShift Container Platform 4.14 or later with OpenShift Virtualization installed, and CloudCasa Self-Hosted server 1.6.x or later. Specific version requirements are noted where they affect design decisions.

Red Hat OpenShift and Virtualization

VMware vs. Red Hat OpenShift Virtualization

OpenShift Virtualization uses KVM (Kernel-based Virtual Machine), QEMU (Quick Emulator), libvirt (virtualization libraries), and CDI (Containerized Data Importer) — the same stack that underpins Red Hat Enterprise Linux and Red Hat OpenStack Platform — to run virtual machines as first-class objects inside an OpenShift cluster. Each VM runs inside a pod (virt-launcher) and is described by a VirtualMachine custom resource. The VM disks are Persistent Volume Claims, not files on a datastore.

VMs share the cluster scheduler, network plane (OVN-Kubernetes plus Multus), and storage plane (CSI) with containerized workloads. This means a single OpenShift cluster can run Linux and Windows VMs alongside container-native applications, and live migration is available when storage supports ReadWriteMany (RWX) block volumes.

vSphere to OpenShift Virtualization Concept Mapping

Concept	vSphere	OpenShift Virtualization
Where VM disks are stored	Datastore	(Persistent Volume Claim) PVC / (Persistent Volume) PV

Policy-based storage	Storage Policy Based Management (SPBM)	StorageClass
Non-disruptive VM migration	vMotion	Live migration (requires RWX) – applies to migration internal to the OpenShift cluster
Active resource balancing	Distributed Resource Scheduler (DRS)	Pod eviction policy, descheduler
Overlay network	NSX-T	OVN-Kubernetes, CNI partners, Multus
Host/VM metrics	vCenter, vRealize Operations	OpenShift Metrics and Monitoring

Architecture / Diagrams

Component Architecture

CloudCasa functionality can be provided in two separate offerings:

1. CloudCasa Software-As-A-Service
2. CloudCasa Self-Hosted

Depending upon organizational requirements, users may opt to go with either model which might have minor variations with overall deployment steps, license enforcement, login authentication and network connectivity, without compromising any core functionality which will be consistent across both models.

CloudCasa Self-Hosted Overview

CloudCasa Self-Hosted is the on-premises edition of the CloudCasa platform. It is feature-equivalent to the SaaS edition but runs entirely on customer-owned OpenShift or Kubernetes, with a customer-owned database and customer-owned storage. Both platforms add an enterprise control plane, multi-cluster catalog, RBAC, scheduling, reporting, and a hardened UI on top.

CloudCasa Self-Hosted is the right fit when one or more of the following applies:

- The cluster runs in an air-gapped or restricted network with no path to the public internet.
- Regulation or sovereignty rules require all metadata and data to stay on the customer's infrastructure.
- The customer operates a service provider or MSP business and needs to host the control plane themselves.
- The customer wants to consolidate authentication, certificate, and registry policy under a single internal Identity Provider (IdP) and Public Key Infrastructure (PKI).

CloudCasa Server Architecture

CloudCasa server runs as a Helm-installed Kubernetes application in a dedicated namespace, typically cloudcasa-server, with the catalog database in a separate cloudcasa-mongo namespace. The server can also be deployed and lifecycle-managed by ArgoCD.

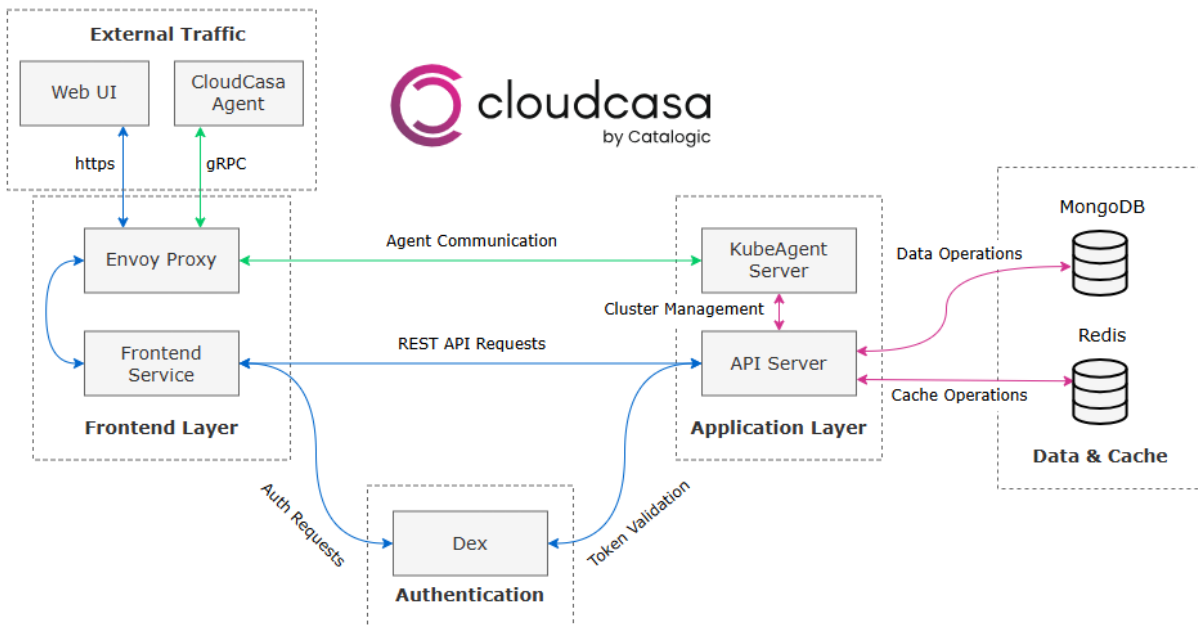


FIGURE 1. CLOUDCASA SELF-HOSTED SERVER LOGICAL COMPONENTS AND EXTERNAL DEPENDENCIES

The CloudCasa server is divided into several major component pods. The major components and their roles are:

Component	Role
-----------	------

amds-envoy	Single TLS-terminating ingress for the server. Exposed via OpenShift Route, NodePort, LoadBalancer, or Istio ingress.
amds-frontend	Web UI.
apiserver	REST API and orchestration. 3 replicas by default; HPA available (2–10 replicas, 80% target CPU).
kubeagentserver (kas)	Agent-side gRPC endpoint. Long-lived TLS connections from cluster agents on port 443. 2 replicas, HPA available.
grpcapiserver	Internal gRPC façade.
Dex	Identity broker. Federates to LDAP, OIDC, AWS Cognito, Google OAuth, or Azure OAuth.
eventreconciler	Reconciles cluster and job events. 3 replicas.
gandalf	Internal routing service.
Redis	Cache.
Minio	Internal object bucket for server logs (not for backup data).
amds-install-helper	Post-install and post-upgrade jobs.
Job runner pods	Short-lived pods spawned per backup or restore action. Few resources, run only while a job is active.
Cloud task pods	Run repository maintenance (expired backup pruning) once per day per object store.
MongoDB	Catalog database in namespace cloudcasa-mongo. Default: 16 GB block-backed PVC. Optionally external (URI + auth secret).

CloudCasa's container images — for the server components (apiserver, kubeagentserver, dex, frontend, and the rest) as well as the per-cluster agent — are pulled from CloudCasa's public registries by default. Users will require an account username and password to access these

images repositories. For self-hosted, air-gapped, or otherwise restricted environments, both image sets can be mirrored into a customer-owned internal registry: the server-side Helm chart is installed with the `images.registry` value pointed at that internal path, while `images.agentRegistry` is set to null to disable the default public pull and instead have agents pull from the same mirrored registry. This mirroring is a required step in the documented air-gapped install flow — download the Helm chart .tgz and full image list from the CloudCasa support portal, push every image into the internal registry, then install or ArgoCD-sync using the local chart — so that no part of the deployment, server or agent, needs a path to the public internet.

CloudCasa Agents on Protected Clusters

CloudCasa Agent Installation

The default method for installing the CloudCasa agent is via a `kubectl` command against a unique manifest containing the `ClusterID` assigned by CloudCasa server described below.

Default method: `kubectl`

1. In the CloudCasa UI, add/register the cluster — this generates a command with a custom manifest that includes a unique `ClusterID`.
2. Copy that command and run it on a host with `kubeconfig` pointed at the target cluster.
3. Requires network access to `api.cloudcasa.io:443` from that host to fetch the agent YAML.
4. Optionally set advanced options first (private container registry, image pull secret) if you need agent images pulled from a private registry.

To uninstall:

```
kubectl delete namespace/cloudcasa-io clusterrolebinding/cloudcasa-io
kubectl delete crds -l component=kubeagent-backup-helper
```

Other installation methods depending on your platform: OpenShift Helm chart or via the OpenShift OperatorHub.

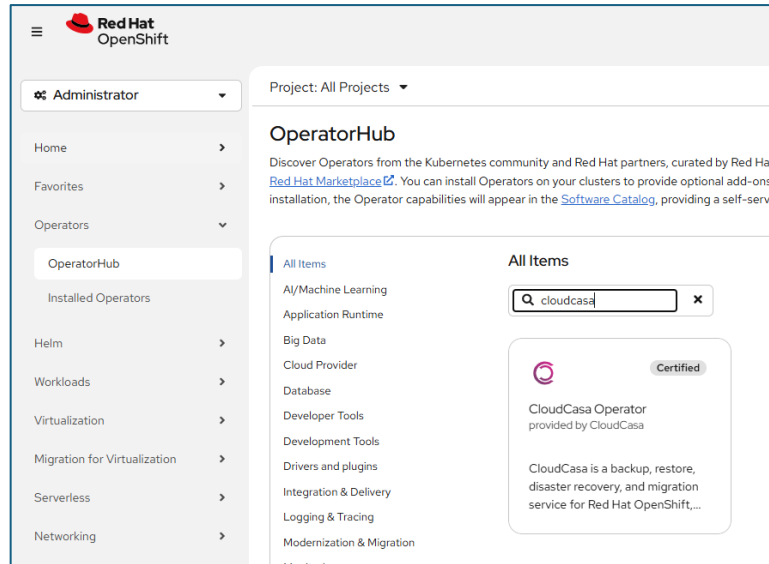


FIGURE 1 OPENSIFT OPERATORHUB

Upon installing the CloudCasa server, you will need to deploy CloudCasa agents for each cluster you are looking to protect. Each protected cluster runs the CloudCasa agent in the cloudcasa-io namespace. The agent provides full feature coverage, including OCP-V VM protection, cross-cluster restore from shared storage, and VM file-level restore.

Browse cluster resources

☐ Namespaces ☐ PVCs ☒ VMs

☐	Namespace ▾	Name ▾	⋮
☐	cloudcasa-sales-demos	centos-stream9-ivory-hare-76	
☐	openshift-test	example	
☐	default	rhel9-fuchsia-puffin-22	
☒	test-openshift	rhel9-ps-test-ocp	
☒	restsmbtest2	rhel9-salmon-bat-76	
☐	default	rhel9-yellow-boa-54	
☐	defaultrestored-smb	rhel9-yellow-boa-54	
☐	windows-vm-namespace	win2k22-prs-block-dynamics	
☐	windows-vm-namespace	win2k22-prs-dynamics	

Items per page: 10 ▲

Cancel **Confirm**

FIGURE 2 SCREENSHOT SHOWING VM SELECTION FROM OPENSIFT VIRTUALIZATION

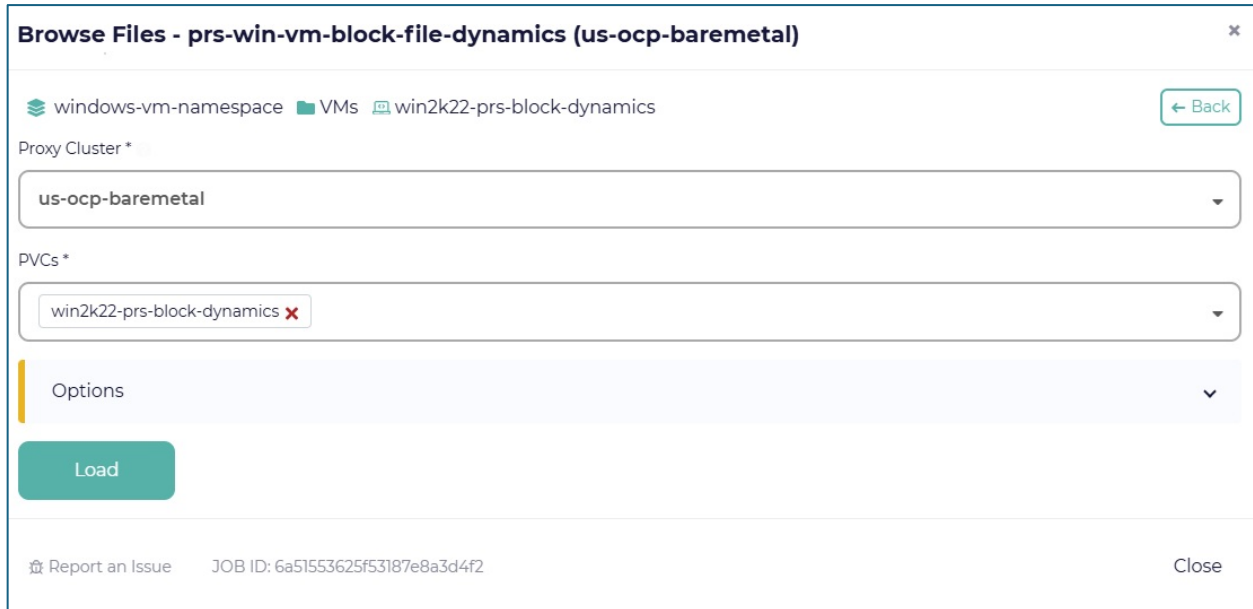


FIGURE 3 SCREENSHOT SHOWING BROWSING FILES ON VM PVC

Network Flow

The traffic pattern is deliberate: only control-plane traffic crosses the server. Backup and restore data flows directly from agents to object / file destination storage and never traverses the CloudCasa server.

- Agents → CloudCasa server on TCP 443 (TLS). Long-lived connection. Used for policy distribution, job orchestration, and status.
- Browser, casactl CLI → CloudCasa server on TCP 443. REST and UI.
- Agents → object / file storage (S3 / S3-compatible / Azure Blob / NFS / SMB). Direct path. The CloudCasa server is not on this path.
- CloudCasa server → SMTP for notifications (outbound).
- dex → external auth provider (LDAP/LDAPS, OIDC, AWS Cognito, Google, Azure).

The platform supports air-gapped installation, where Helm charts and container images are mirrored to an internal registry.

Data Movement (File Index / CSI Snapshots)

CloudCasa's data mover pod is a short-lived worker that CloudCasa spins up per backup, restore, replication, or migration job to move data off (or onto) a PVC; it isn't a long-running part of the server, and it disappears once the job finishes. It comes up on the protected workload's own cluster, in the same namespace as the source PVC (as seen in the cloudcasa-io-data-mover pod

label used to scope network policy for it), rather than inside the CloudCasa server's own namespaces — keeping the actual data path local to the workload rather than routing through the control plane.

Job definitions select what to back up by namespace, by label selector, or by cluster-scope rules. Job types are:

- Snapshot-based backup: CSI VolumeSnapshot for PVCs, plus capture of all selected Kubernetes resources. This is the primary mechanism for OCP-V VMs because it offers the cleanest, fastest point-in-time recovery.
- Live backup: for PVCs whose CSI driver does not support snapshots, or where filesystem-level backup is sufficient.

Protect PVs using:

☐ Snapshot only
 ☒ Copy to backup storage

☐ Snapshot PVs before backup where possible
 ☐ Back up directly from all PVs not snapshotted
 ☒ Configure backup method by storage class (Advanced)

lvms-vg1	Read data from snapshot
nfs-rwx	Read data from PVC
ocs-external-storagecluster-ceph-rbd	Read data from PVC Read data from underlying host volume Skip Read data from snapshot
ocs-external-storagecluster-cephfs	Read data from snapshot
openshift-storage.noobaa.io	Read data from snapshot
smbbaremetal	Read data from snapshot

FIGURE 4 SCREENSHOT ON HOW TO CONFIGURE LIVEBACKUP OPTIONS AT THE STORAGE CLASS LEVEL

☒ Configure data mover node affinity ?

☒ By storage class

☐ For all storage classes

lvms-vg1	Auto
nfs-rwx	Auto None PVC Node Not PVC Node
ocs-external-storagecluster-ceph-rbd	
ocs-external-storagecluster-cephfs	Auto
openshift-storage.noobaa.io	Auto

FIGURE 6 CONFIGURE NODE AFFINITY THROUGH WHICH PVC WILL BE PROTECTED

Storage Topology

Source PVC/PV (Block Ceph / File NFS)

Storage Prerequisites for VM Workloads

OpenShift Virtualization replaces shared datastores with per-VM PVCs. For CloudCasa to protect VM workloads on OpenShift, the underlying CSI driver must offer:

- VolumeSnapshot support for storage-level point-in-time snapshots.
- Read-Write-Many (RWX) block VolumeMode if you require live migration during a backup window.
- A correctly annotated VolumeSnapshotClass so CloudCasa can drive snapshot operations.

Common validated storage choices may include: Red Hat Open Data Foundation (Ceph RBD), HPE Alletra, Dell, IBM Storage, NetApp, *and more*.

Storage Validation

Before pointing CloudCasa at production VM workloads, validate that the StorageClass and VolumeSnapshotClass are correctly configured. The CloudCasa server can be configured to verify CSI snapshot and restore capabilities as part of cluster onboarding. A minimal validation procedure is:

1. Verify the StorageClass has the correct provisioner and access modes (RWX, block VolumeMode for OCP-V live migration).
2. Verify a matching VolumeSnapshotClass exists and is annotated as the default for the driver.
3. Create a small PVC, take a CSI snapshot, restore the snapshot to a new PVC, and verify the contents.
4. Repeat with a 50 GiB-ish VM disk PVC backing a running OCP-V VM and verify live migration works during a backup.

For further details on manually verifying your CSI configuration you can reference the following link: [Verifying CSI configuration - CloudCasa User's Guide](#)

Backup Storage Locations

CloudCasa backs up to object storage, supporting any S3-compatible or Azure Blob storage, including private/isolated endpoints, with optional immutability. It also supports NFS and SMB as file-level destinations.

Backup Storage 							
① Default Storage: CloudCasa Storage (azure: centralus) 							
Object Storage				File Storage			
Name 				 Filter 		<button>Add Storage +</button>	
Name 	Provider 	Bucket Name 	Region	Endpoint	Status	Cluster	
Akamai S3	s3	mcp-cloudcasa-akamai	us-east-1	https://us-southeast-1.linodeobjects.com			Actions 
BackBlaze-S3 Public-ObjectLock-Enabled 	backblaze	test-objectlock2	us-east-1	https://S3.us-west-004.backblazeb2.com			Actions 
Backblaze-S3 Public	backblaze	cloudcasa-s3-objectlock-disabled	us-east-1	https://S3.us-west-004.backblazeb2.com			Actions 
CN-Cloudian S3	s3	catalogic		https://S3.us-west-1.cloudianhyperstore.com:443			Actions 
Hitachi Vantara S3 	s3	cloudcasa		https://partner.contentplatform.hitachivantara.com		us-ocp-baremetal	Actions 

FIGURE 5 BRING YOUR OWN OBJECT / FILE STORAGE TO BE USED AS A BACKUP DESTINATION

Encryption

All communications between the agent and the CloudCasa Server are encrypted using TLS. All communication with object storage is always encrypted using TLS. CloudCasa encrypts data in flight (TLS, end-to-end between agent and object store) and at rest. For File backup storage locations, CloudCasa writes data to NFS/SMB PVC in encrypted form. So irrespective of how data is transferred between the NFS client and server, it is encrypted at rest by CloudCasa.

Object Lock and Immutable Backups

CloudCasa supports S3 Object Lock on user-managed S3-compatible storage and Azure Blob immutability policies. Configure the bucket with the desired retention (governance or compliance mode) before pointing CloudCasa at it. Once written, backup objects cannot be deleted or overwritten until the retention expires — including by an administrator or by a compromised CloudCasa service account.

Validated immutable targets include:

- MinIO with Object Lock enabled.
- Red Hat Ceph Object Gateway (RGW) with Object Lock.
- Dell ECS, NetApp StorageGRID, Hitachi HCP, HPE Alletra Storage MP X10000

The DR passphrase (or, more generally, the credentials and keys CloudCasa uses to write to the object store) are a high-value target for ransomware. Treat them accordingly: store in a sealed secret backend (HashiCorp Vault, SealedSecrets, External Secrets Operator), rotate on a schedule, and monitor for any access outside the CloudCasa service account.

Functionality

VM Backup / Restore

CloudCasa protects virtual machines running on Red Hat OpenShift Virtualization using the same unified interface it uses for standard Red Hat OpenShift application backups. VM detection is automatic, so no special configuration is needed: users can back up an entire cluster (the recommended approach) or select individual VMs or namespaces, and CloudCasa will automatically capture all associated resources, including the VirtualMachine and VirtualMachineInstance objects, DataVolumes, PVCs, pods, secrets, and ConfigMaps. Where the QEMU Guest Agent is installed inside a VM, CloudCasa uses it to freeze and unfreeze the

filesystem during backup for a consistent, crash-safe snapshot; VMs without the guest agent are still backed up, just without that extra consistency guarantee.

Restore works the same way as any other Kubernetes restore, letting you recover a full cluster, specific namespaces, or individual VMs picked out through the resource browser, with all related resources pulled in automatically. CloudCasa adds VM-specific restore options as well, including clearing or preserving MAC addresses, generating a new firmware UUID to avoid identity conflicts with a still-running source VM, and choosing the run strategy for restored VMs (for example, restoring them in a halted state so they can be reviewed before being powered on). CDI and KubeVirt need to already be installed on the target cluster for VM restores to succeed, though this generally isn't a concern for same-cluster restores or clusters already set up for OpenShift Virtualization.

Beyond full VM recovery, CloudCasa also supports file-level restore from VM backups: since VM disks are opaque block volumes, CloudCasa mounts an image of the VM's filesystem through a proxy VM automatically created by CloudCasa on the OpenShift Virtualization cluster so individual files can be browsed and downloaded without restoring the whole VM. Combined with cross-cluster and cross-cloud recovery, encryption, and retention policies, this gives CloudCasa a complete story for protecting and migrating VM workloads alongside regular containerized applications, all from one console.

Browse cluster resources

Search

Namespaces

PVCs

VMs

☐	Namespace	Name	
☐	cloudcasa-sales-demos	centos-stream9-ivory-hare-76	
☐	openshift-test	example	
☐	default	rhel9-fuchsia-puffin-22	
☒	test-openshift	rhel9-ps-test-ocp	
☒	restsmbtest2	rhel9-salmon-bat-76	
☐	default	rhel9-yellow-boa-54	
☐	defaultrestored-smb	rhel9-yellow-boa-54	
☐	windows-vm-namespace	win2k22-prs-block-dynamics	
☐	windows-vm-namespace	win2k22-prs-dynamics	

Items per page: 10

Cancel

Confirm

VM Protection Scope (OCP-V)

CloudCasa is application-centric: policies target Kubernetes resources (namespaces, labels, or cluster-scoped objects) rather than nodes or hypervisors. The same engine protects containerized applications and OCP-V VMs.

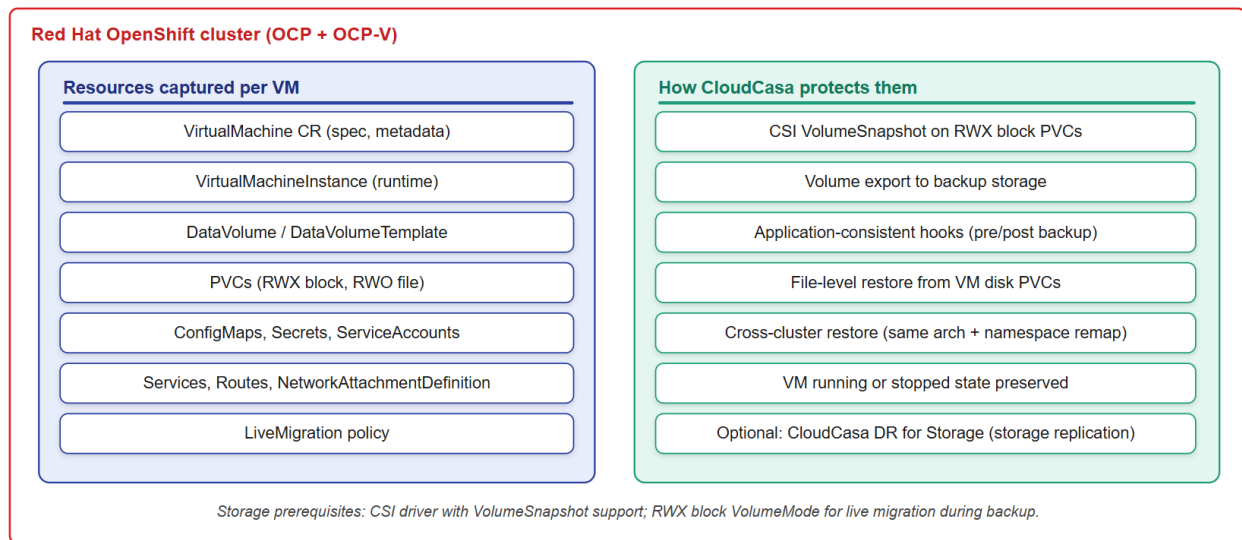


FIGURE 8 WHAT CLOUDCASA CAPTURES WHEN PROTECTING A VM ON OPENSIFT VIRTUALIZATION

Snapshot retention is configured per policy with hourly, daily, weekly, monthly, and yearly tiers. Snapshots are taken on the source storage system, and a copy is exported to the off-cluster backup target on every successful snapshot.

Alternatively, Live Backup is another option one can use in CloudCasa.

Live Backup in CloudCasa

Live Backup is CloudCasa's fallback data-protection method for persistent volumes whose storage driver doesn't support (or fully support) CSI snapshots. Instead of snapshotting the volume first, CloudCasa's data mover pod reads directly from the live PVC (or, in some cases, from the underlying kubelet host path) and streams that data to backup storage. Because there's no point-in-time snapshot involved, consistency isn't automatically guaranteed — CloudCasa recommends pairing Live Backup with application hooks to quiesce workloads (e.g., flushing writes, briefly pausing a database) so the copy isn't taken mid-write.

Policy Examples

A typical production policy when assigned to a backup job definition an OCP-V VM workload looks like:

Item	Value
Name	vm-prod-hourly
Type	Snapshot policy
Selection	By label: app=prod-vm, plus all resources in the workload namespace
Frequency	Hourly
Backup window	20:00 to 04:00 local time, staggered
Retention	24 hourly, 7 daily, 4 weekly, 12 monthly, 7 yearly
Export	Every snapshot exported to backup target location profile
App hooks	Pre-backup: fsfreeze inside guest. Post-backup: fsfreeze --unfreeze. Pre/post-restore: VM stop/start as appropriate.
Destination on restore	Original namespace, original storage class. Transform set available for cross-cluster moves.

Replication and Migration

CloudCasa can also extend its backup engine into two related but distinct capabilities: cluster replication, for keeping a live standby copy of a cluster in sync, and cluster migration, for one-time moves of workloads between clusters, clouds, or on-premises environments. Both features reuse the same underlying data mover, application hooks, and restore-transform logic as CloudCasa's regular backup and restore jobs, but they're configured and scheduled differently to match their purpose.

Note: Replication and Migration workflows do not have the Resource Browser component to browse VM objects and a workflow can be created to replicate or migrate the VM namespace for these specific use cases.

Replication

Replication jobs continuously copy a full cluster, or selected namespaces, to a destination cluster on a recurring schedule. The typical use case is maintaining a DR or hot-standby cluster, or keeping a dev/test environment in sync with production. Like backups, replication jobs run using a policy-driven schedule (the same policy mechanism used for backups) and can also be triggered manually or via the API. One way to envision replication is to think about having a scheduled backup and restore policy running in that exact sequence driven by a policy.

Setting up a replication job follows the same flow as a backup job: both the source and destination clusters need to be registered with CloudCasa. As with backups, you can apply restore transforms (most commonly for changing storage classes or preserving node ports) and pre/post application hooks for workloads like databases that need to be quiesced for consistency.

Migration

Migration jobs perform a one-shot copy of a cluster or part of a cluster, rather than a recurring sync. They're used for moving an application from one cluster to another or moving between on-premises and cloud (in either direction), or for migrating a workload locally to a cluster with a different configuration. Migration jobs can't be scheduled on a policy the way backups and replication jobs can — they run once, either immediately after being defined, or later via manual trigger or API call.

The setup workflow mirrors that for replication jobs: register source and destination clusters choose what to migrate by namespace, resource type, label, or PV inclusion/exclusion, and apply restore transforms and app hooks as needed.

Using a migration workflow will be applicable when performing a one-time move of an application from one OpenShift cluster to another.

Migration Transforms

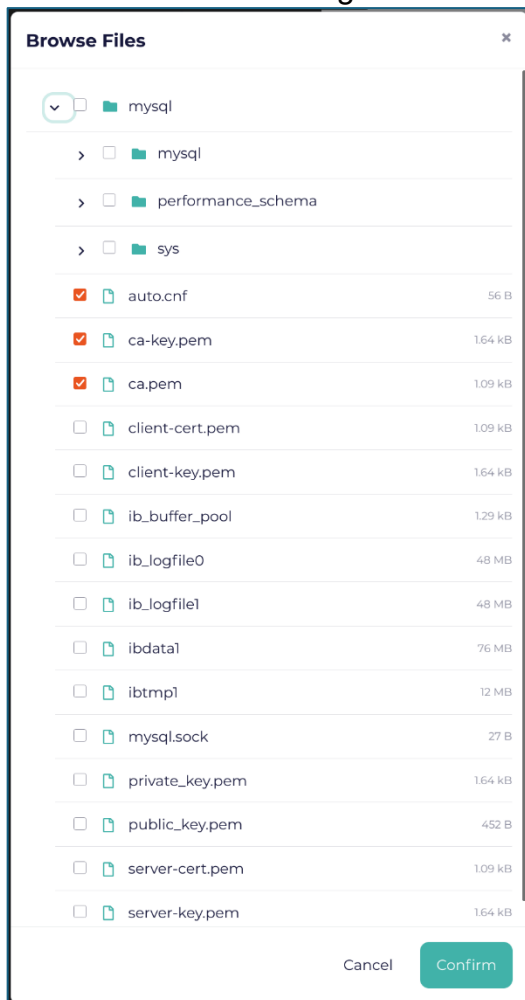
Because CloudCasa captures the application and VM as a Kubernetes resource graph plus its data, restores can be transformed during recovery. This is what makes mobility work: a workload backed up on Cluster A with, for example, HPE storage can be restored on Cluster B with a different CSI driver by remapping the StorageClass. Other common transforms include:

- Namespace rename (production to dev, or tenant remap).
- StorageClass remap (between CSI drivers).
- Replica count adjustment (scale down on restore to a smaller cluster).
- Annotation and label edits (for example, removing production-only operators on a dev restore).

For OCP-V specifically, this enables VM migration between OpenShift clusters with different storage backends — a useful path when consolidating from edge clusters to core, or when refreshing storage hardware under a running VM fleet.

Granular File Recovery

- **File-level restore:** browse the files in any backed-up PVC and restore selected files to an existing or new PVC.



- **Resource Browser navigation:** CloudCasa's Resource Browser lets users drill into a backed-up PVC or VM virtual disk and navigate its file and folder structure directly from the CloudCasa UI, without needing to mount or restore the entire volume first. Users can search or browse to the specific files they need, select one or multiple files (or folders), and initiate a restore of just that selection.

- **Flexible restore targets:** selected files can be restored to their original location, an existing PVC, a new PVC, or even an alternate cluster or namespace — giving administrators the flexibility to recover data into a different environment for testing, migration, or forensic review without disrupting the original workload.
- **VM use case:** for VM workloads this means a user can browse the OpenShift Virtualization VM filesystem to pull back an accidentally deleted config file or document without performing a full VM restore — saving time and avoiding unnecessary downtime.

Application Support

Application consistency is delivered via app hooks: pre-backup, post-backup, and post-restore commands that run inside selected pods. Common app hook patterns include:

- Database freeze/thaw (MongoDB, PostgreSQL, MySQL, Oracle, etc.).
- fsfreeze on the guest filesystem before snapshot for OCP-V VMs (automatic if QEMU guest agent is present).
- etcd snapshot via the deploy-cloudcasa-openshift-etcd-backup-runner pod for OpenShift control plane recovery.

For application-consistent database backups (for example, a stateful MongoDB or PostgreSQL backing an app namespace):

Item	Value
Name	app-prod-hourly
Type	Snapshot policy
Selection	By name: app-prod namespace, all resources
Frequency	Hourly
Retention	24 hourly, 7 daily, 4 weekly, 12 monthly, 7 yearly
Export	Every snapshot to Object or File Storage Destination
App hooks	Pre-backup: <code>db.fsyncLock()</code> / <code>pg_start_backup()</code> . Post-backup: <code>db.fsyncUnlock()</code> / <code>pg_stop_backup()</code> .

Destination on restore	By default the source namespace. Restore wizard allows transforms (namespace, storage class, replicas).
------------------------	---

FIGURE 9 APPLICATION AWARE BACKUPS THROUGH PRE/POST APP HOOKS

Disaster Recovery

Introduction

The CloudCasa DR feature pairs its OpenShift-aware orchestration with native remote replication on supported storage arrays, rather than relying solely on backup data, to enable fast, low-RTO failover. It currently supports HPE Alletra MP B10000 and 9000 storage arrays, Dell PowerStore, with array-side replication handling the data layer while CloudCasa automates failover and failback, namespace-level recovery, and multi-tenant isolation across clusters and sites. This lets organizations recover both OpenShift container applications and OpenShift Virtualization VMs

using the storage hardware they already have, rather than being limited to restores coming solely from backup storage.

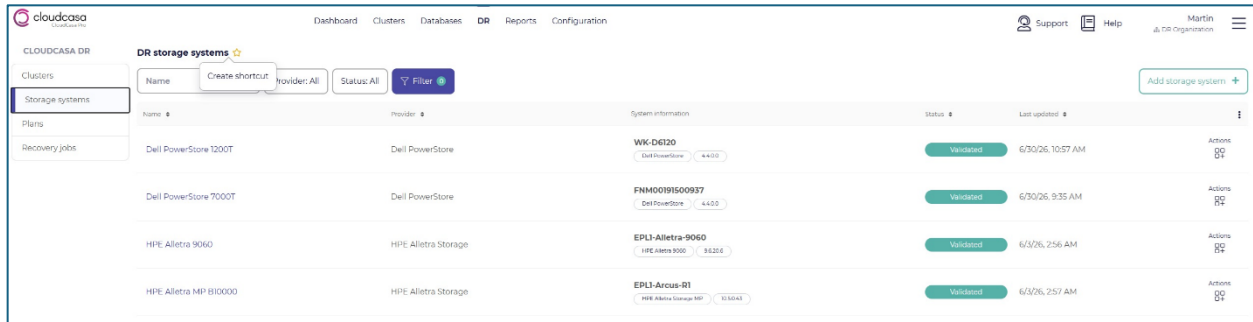


FIGURE 10 PLAN YOUR DISASTER RECOVERY BY ADDING STORAGE SYSTEMS TO CLOUDCASA

With CloudCasa DR in place and one or more DR Plans in place for a cluster, CloudCasa DR can be used to quickly fail over workloads from the source cluster to the target cluster in the event of a disaster. (For a guided tour through the setup process, see the [CloudCasa DR Setup Guide](#).) Failover is accomplished using DR Recovery jobs. These can be defined in advance so that they can be initiated quickly when needed.

Inventory PVC

Unlike a standard CloudCasa backup which protects the resource YAML definitions by backing them up to S3 object or file storage, CloudCasa uses the concept of an Inventory PVC, which is a persistent volume created on the storage array and mounted to the source cluster. The inventory PVC stores those resource definitions required during CloudCasa DR so that users will not be required to store this information in a separate backup location (like S3 object storage) when a disaster recovery performed. The inventory PVC (and persistent volume) will also be replicated through storage replication, thereby ensuring that it can then be mounted at the destination cluster to perform a recovery of those resources as defined within the disaster recovery plan.

Failover

If there is a problem with the primary site or if you want to migrate the workloads to the DR site, you will need to perform a failover. This is achieved by creating and running a “DR Recovery” job.

What happens during failover

During failover, CloudCasa validates that disaster recovery can proceed and verifies that no resources on the target site would block successful recovery.

CloudCasa begins by synchronizing the consistency group associated with the inventory PVC. If synchronization fails, CloudCasa continues with the failover process. It then invokes the storage system to fail over the consistency group.

When the consistency group failover is complete, CloudCasa restores the PVCs, PVs, and any required Custom Resources. It waits for the PVCs to reach the Bound state, then starts the DR Agent on the new primary site. After the DR Agent starts, CloudCasa repeats the same workflow for the workloads selected during DR Recovery job creation: synchronization, failover, and volume restoration. At the end of the recovery, the workloads should be running on the target cluster.

An important point to note is that once a failover is done, the corresponding DR Plan and DR Recovery job cannot be used anymore. As a result, if you only want to failover a over a subset of the cluster's workloads, it is recommended that you create a separate DR plan for only those workloads as you will not be able to select subset of resources to recover from the original plan. You can create a new plan from scratch or clone the existing DR plan (using **Actions -> Clone**) and select new workloads and a new Inventory PVC.

Pre-failover diagram

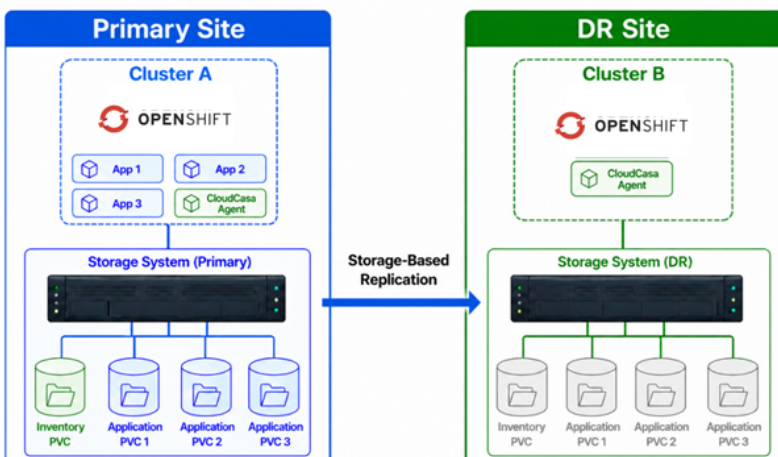


FIGURE 11 PRE-FAILOVER DIAGRAM

Post-failover diagram

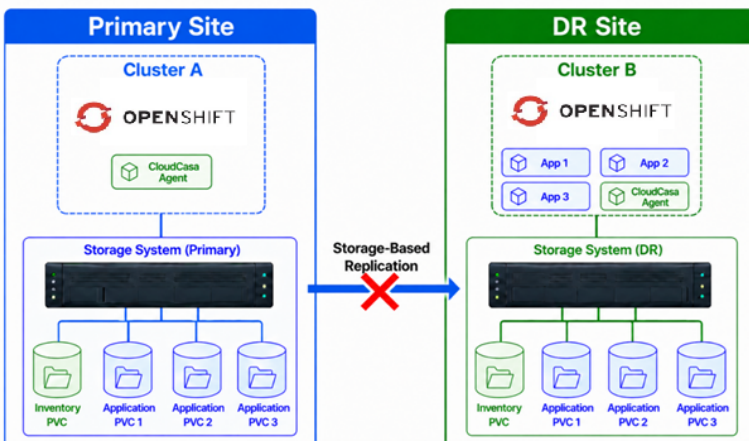


FIGURE 12 POST-FAILOVER DIAGRAM

Failback

Failback is a disaster recovery operation that moves workloads from the DR site back to the Primary site. Users must create the appropriate DR Plans and DR Recovery jobs before initiating failback.

What happens during failback

In supported storage array environments, the CloudCasa DR failback workflow depends on the current state of the discovered consistency groups or replication groups. When replication is operating in the reverse direction following a failover, CloudCasa initiates the appropriate storage platform operation to reverse the replication direction and synchronize any changed data. After synchronization is complete, CloudCasa returns the replication groups to their original, pre-failover source and target roles.

After the storage-side operations are complete, CloudCasa restores the inventory volume on the Primary site first. Once the inventory volume is restored and the DR agent process starts, it restores the Kubernetes resources and their corresponding volumes.

If the discovered consistency groups are already in the “Primary” state, CloudCasa uses the same process described in the failover section.

More information can be found here: [CloudCasa DR Failover Guide - CloudCasa User's Guide](#)

Disaster Recovery of the CloudCasa Server

While normally resolved through a Catalog Restore operation, for those instances in which the cluster is enabled for disaster recovery and using a supported storage appliance, one can accomplish a DR of the CloudCasa server and corresponding services for workflows and DR

plans that have been defined within the DR tab. These two-server configurations would be useful when there is the potential for bi-directional failover configuration (i.e. where there are two sites with production instances of the CloudCasa running at both sites). In a DR scenario site A would failover to site B, or site B would failover to site A.

Note: CloudCasa catalog restores will be required when recovering backup / restore workflows, cluster definitions, storage destinations, etc. (i.e. configuration outside of the DR workflow tab)

In a typical DR configuration, workloads would be running in a cluster at the primary site while the cluster at the DR site is in a standby state, ready to start running workloads if the primary site fails. In this scenario, one may want to install two different CloudCasa servers - one at primary site and the other at DR site. The primary cluster will be registered in the CloudCasa server running in the primary site while the DR cluster will be registered in the CloudCasa server running at DR site. To be clear, installing two CloudCasa servers as described above is not mandatory. CloudCasa fully supports using the same CloudCasa instance for both primary and DR clusters.

In the two-server architecture, the CloudCasa server running at the primary site is referred to as the **primary CloudCasa server**, and the CloudCasa server running at the DR site is referred to as the **remote CloudCasa server**.

Adding a remote CloudCasa server

A remote CloudCasa server can be added to CloudCasa by going to **DR => CC servers** and clicking on **Add CC server** and supply the requested information for your CloudCasa server.

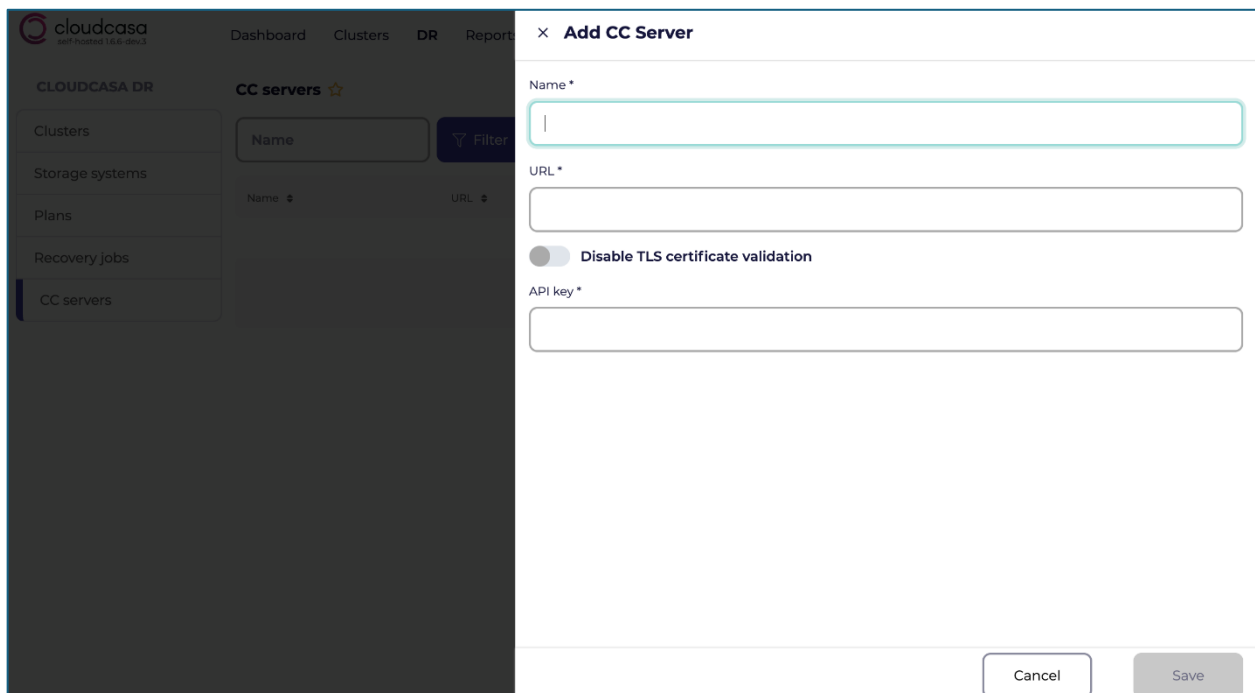


FIGURE 13 ADD REMOTE CLOUDCASA SERVER FOR DR

Automatic reverse configuration

When a remote CloudCasa server resource is added, CloudCasa automatically creates a matching CloudCasa server resource on the remote CloudCasa server that points back to the primary CloudCasa server. This includes generating the required API key on the remote CloudCasa server. The reverse resource is named using the pattern:

Reverse CloudCasa server (<NAME-OF-THE-CREATED-CloudCasa-SERVER>)

This automatic reverse configuration enables requests to be initiated in either direction, which is recommended for keeping resources in sync and also for failover/failback operations. Bidirectional reachability is not strictly required for failover: if the primary site is down, failover still works, but workloads are not stopped automatically on the primary site. DR recovery jobs themselves are managed from the remote CloudCasa server.

Network connectivity requirements

The two CloudCasa servers communicate by sending authenticated REST requests to each other over HTTPS.

It is highly recommended that both CloudCasa servers use a stable DNS name so that even if a server needs to be reinstalled (and catalog restored), it can still be reached by the other CloudCasa instance.

Using the remote CloudCasa server in a DR plan

While creating a **DR Plan**, users can configure a CloudCasa server, if one is present.

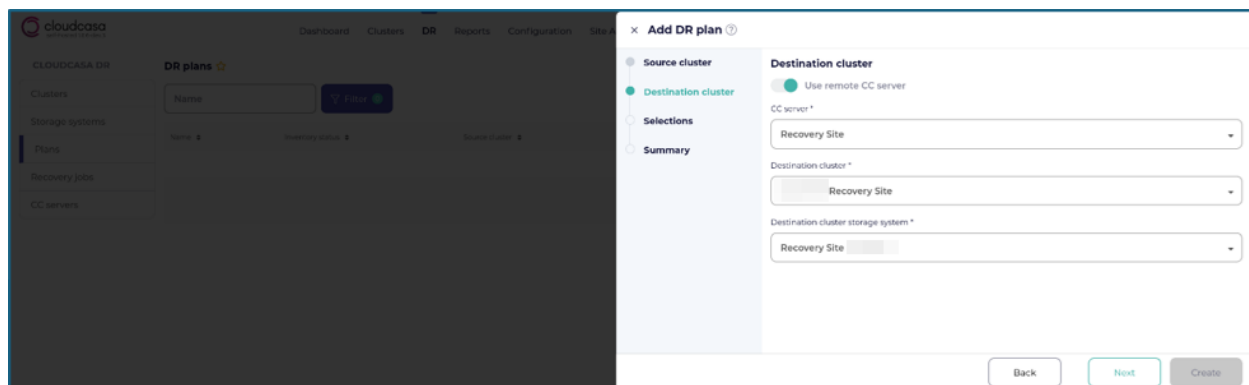


FIGURE 14 CONFIGURE DR PLAN FOR REMOTE CLOUDCASA SERVER WHEN A REMOTE CLOUDCASA SERVER IS CONFIGURED FOR A DR PLAN, SEVERAL DR-RELATED RESOURCES, SUCH AS THE DR PLAN, STORAGE SYSTEMS, CLUSTER STORAGE SYSTEMS, AND STORAGE

VOLUMES, ARE SYNCED FROM THE PRIMARY CLOUDCASA SERVER TO THE REMOTE CLOUDCASA SERVER. THE SYNCED RESOURCES ON THE REMOTE CLOUDCASA SERVER ARE READ-ONLY. IN ADDITION, DR RECOVERY JOBS CAN ONLY BE CREATED AND RUN FROM THE REMOTE CLOUDCASA SERVER.

More information can be found here: [DR CloudCasa Servers - CloudCasa User's Guide](#)

Recommended Scenarios

Two Control Plane Topologies

CloudCasa Self-Hosted supports two server placement patterns. Both protect the same set of workloads with the same feature set. The choice is driven by failure-domain separation, cluster-count economics, and operational ownership.

Pattern A: Dedicated Management Cluster (Recommended for Production)

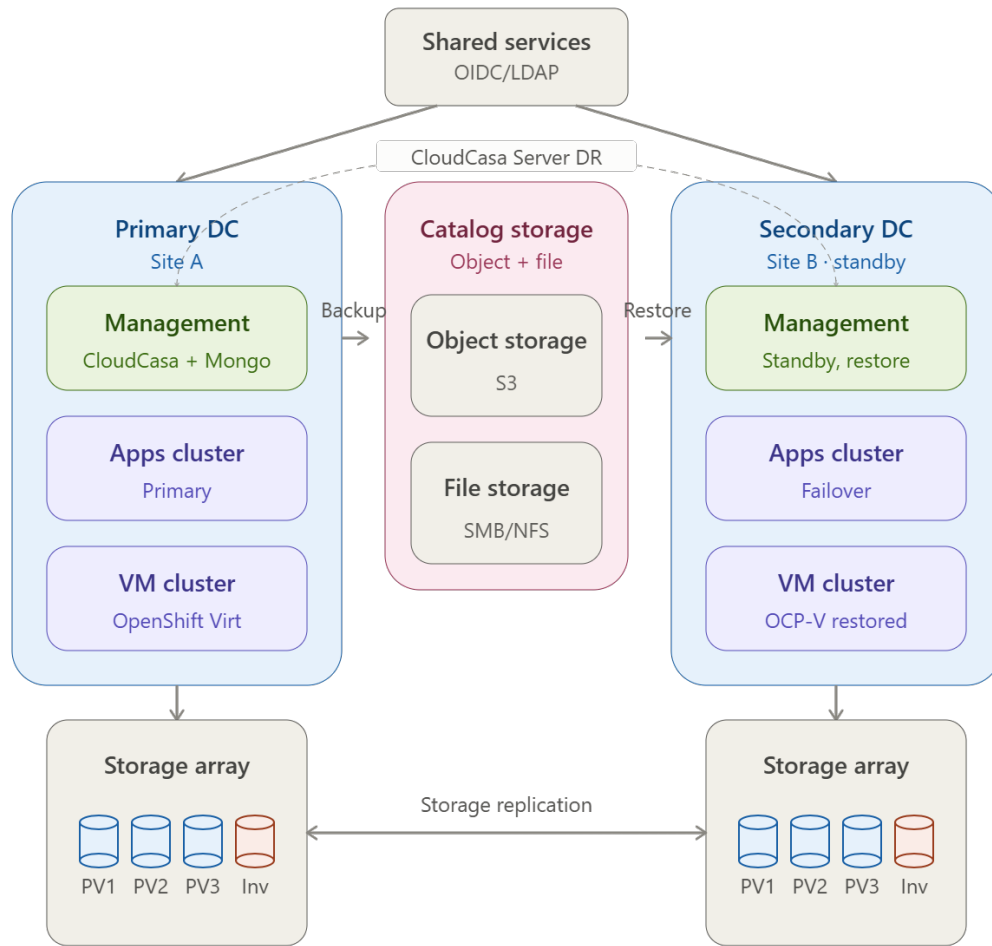


FIGURE 15 RECOMMENDED ARCHITECTURE: DEDICATED CLOUDCASA MANAGEMENT CLUSTER WITH ON-PREMISES PRIMARY PLUS ON-PREMISES SECONDARY LOCATED IN ALTERNATE DATACENTER.

NOTE HERE THAT THE CLOUDCASA MANAGEMENT SERVER (IN GREEN) WILL BE INSTALLED ON ITS OWN CLUSTER WITH CONTAINERIZED WORKLOADS AND VMS SCHEDULED ON OTHER OPENSIFT CLUSTERS

In Pattern A, the CloudCasa server runs on a dedicated OpenShift cluster that is operationally separate from the workload clusters it protects. A second CloudCasa management cluster is deployed at the secondary site and configured as a hot standby. During a DR event, you run the

CloudCasa DR recovery workflow against the standby server to fail over and spin up protected clusters at the DR site — this is the primary recovery action, not a catalog restore. A catalog restore is only needed as an additional step if specific backup workflows themselves must be recovered at the DR site, to bring in the backup metadata required to run restores there — and when it's needed, it follows the DR failover rather than replacing it. This pattern matches the operational model of traditional enterprise backup infrastructure: the data protection control plane is its own failure domain, with its own failover path.

Pick this pattern when:

- There are multiple workload clusters to protect.
- The data protection function is owned by a separate team from the application clusters.
- You want catalog and configuration to survive a complete loss of any workload cluster.
- Air-gap or compliance requires the protection plane to be isolated from production workload reach.

Pattern B: Co-Located on the Workload Cluster

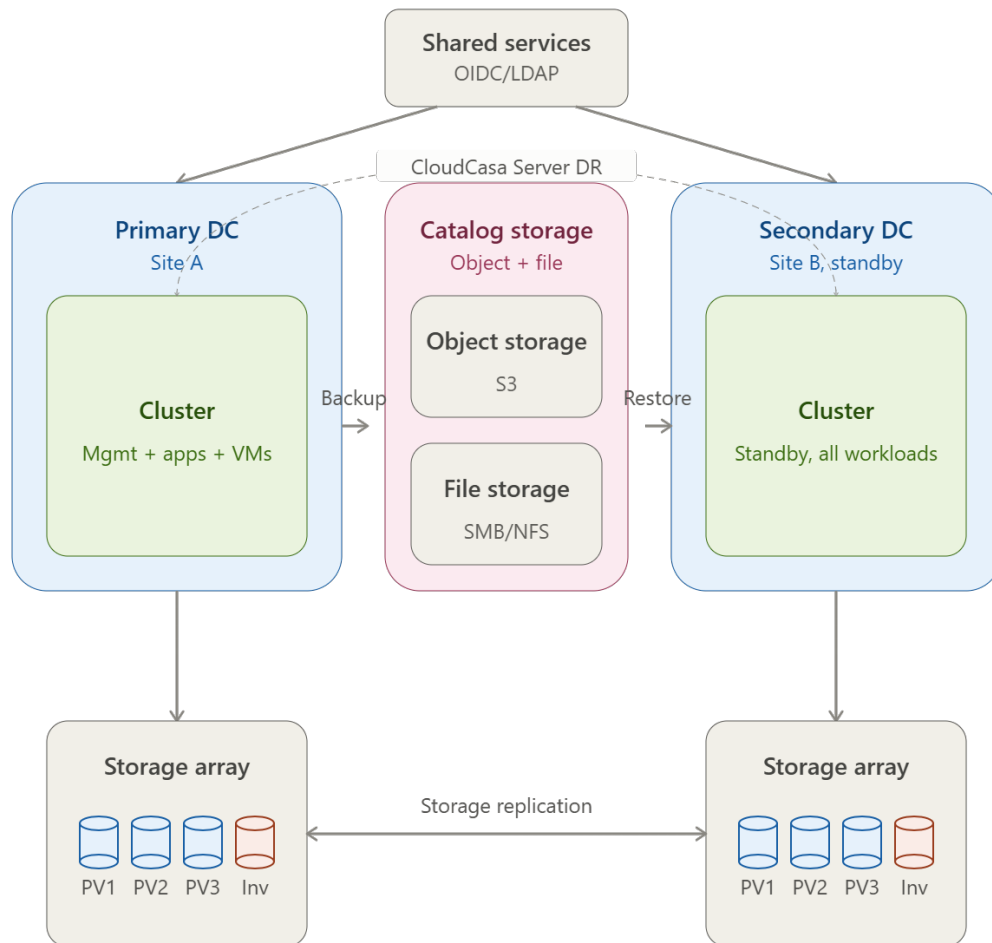


FIGURE 16 CO-LOCATED VARIANT: CLOUDCASA SERVER RUNNING ON THE SAME OPENSIFT CLUSTER AS THE WORKLOADS IT PROTECTS

In Pattern B, the CloudCasa server runs in its own namespaces on the same OpenShift cluster that hosts workloads and OCP-V VMs. This reduces the footprint to one cluster but binds the protection plane to the same failure domain as the workloads it protects. The distinction from Pattern A is purely about where the CloudCasa server itself lives — dedicated cluster versus shared cluster — not about how backup or catalog data is protected, which works the same way in both patterns.

Pick this pattern when:

- There is exactly one workload cluster, or a small lab / edge installation.
- The cost of a dedicated management cluster is hard to justify.
- Operational ownership of clusters and protection sits with the same team.

Catalog Protection and DR Recovery (applies to both patterns)

Regardless of the pattern, CloudCasa must protect its own catalog by exporting catalog backups to off-cluster object or file storage, since the CloudCasa server cluster cannot back itself up.

Whether a catalog restore is required during DR depends on whether a standby CloudCasa server is available at the DR site — not on which pattern you deployed:

- If a standby CloudCasa server is running at the DR site (Pattern A configured as a hot standby), run the CloudCasa DR recovery workflow first to fail over and spin up clusters at the DR site. A catalog restore is only needed after that if specific backup workflows need to be recovered at the DR site, to pull in the backup metadata required to run restores there.
- If no standby server is available at the DR site (typically Pattern B, or Pattern A without a configured standby), recovery requires deploying a new CloudCasa server — typically at the secondary site — and running `casactl` to perform a catalog restore from the backup storage location (BSL), rebuilding state from the off-cluster catalog backup before any restores can run.

For most production environments already using storage replication, pairing it with a CloudCasa hot-standby server (Pattern A) is recommended: the two servers' catalogs stay continuously synchronized through the inventory PVC, which is what delivers near-zero RPO/RTO during failover — rather than depending on a catalog restore at DR time.

Recommended Production Layout

The recommended pattern for production deployments is Pattern A: a dedicated CloudCasa management cluster at the primary site, a warm-standby CloudCasa management cluster at the secondary site, and one or more OpenShift workload clusters at each site running OCP-V workloads. Catalog backups run hourly to the off-cluster backup storage location, which is hosted in a separate failure domain — typically a different building or a different power and cooling domain on the same campus. If an S3-compatible object storage provider is being used, then Object Lock can be enabled on the bucket for immutable retention.

Authentication for both management clusters federates to the same external OIDC or LDAP service — for example, Okta, Microsoft Entra ID via OIDC, or an on-premises Keycloak. This

ensures both clusters share an identity directory that is outside the failure domain of either workload cluster.

Primary and Secondary Site Layout

At the primary site:

- One OpenShift management cluster (3 master nodes plus enough worker nodes for the CloudCasa server).
- One or more OpenShift workload clusters running OCP-V VMs and containerized apps.
- CSI-capable deployed on the workload cluster(s), with block volumeMode and CSI snapshots enabled.
- CloudCasa agent installed in the cloudcasa-io namespace on each workload cluster.

At the secondary site:

- Mirror of the workload cluster topology, sized to take the recovered workload set.
- A standby CloudCasa management cluster is recommended, but not required. You can also instantiate a new CloudCasa server and restore the catalog from its backup storage location during a DR event.
- Independent object / file storage path (the backup destination of the CloudCasa catalog should be reachable from both sites).

Permutations and Other Patterns

The architecture above stays inside an on-premises envelope. The patterns below cover two largely independent decisions: where the CloudCasa server itself runs and how it's protected, and how the customer chooses to arrange their own workload clusters across sites. Mixing these two decisions together is a common source of confusion, so they're addressed separately.

CloudCasa Server Placement and Protection

Regardless of which workload topology a customer chooses, the CloudCasa server's own availability follows the same model everywhere:

- CloudCasa can run co-located on a customer's workload cluster (Pattern B) or on its own dedicated cluster. A dedicated cluster is the recommended approach, since it decouples the CloudCasa server's lifecycle and availability from the workloads it protects.
- CloudCasa does support a hot standby configuration for the CloudCasa server itself. There is no live failover of the clusters between the two sites as it must be invoked via the DR Recovery job.

- *Note: CloudCasa server DR can help mitigate the requirements for accommodating a DR workflow should the CloudCasa server on the primary site go down, but if backup and recovery workflows are involved, then a full catalog restore will be required at a DR site.*
- The critical requirement in every deployment is that the CloudCasa catalog database is backed up to a backup storage location that is reachable independent of the cluster CloudCasa runs on. If the CloudCasa server's cluster is lost, recovery consists of provisioning a new cluster (commonly rebuilt from the same GitOps repo) and restoring the catalog from that backup storage location — on-site or off-site as needed.
- Whether a customer additionally maintains a warm standby cluster ready to receive that catalog restore is a business continuity decision, not a CloudCasa requirement. Its value depends entirely on how quickly the customer can otherwise stand up a replacement cluster; it doesn't change how CloudCasa itself operates or is protected.

Workload Cluster Topologies

These patterns describe how customers arrange the clusters running their own applications — independent of where the CloudCasa server sits.

- **Single-Site, On-Premises Only:** Smallest viable production deployment — one OpenShift workload cluster and an off-cluster backup storage location reachable over the network from the cluster. The cluster cannot back itself up, so recovery depends on that off-cluster storage backup storage location and on the ability to provision a replacement cluster to restore into.
- **Service Provider / MSP Variant:** Multi-tenant deployments run a single CloudCasa Self-Hosted instance with multiple organizations defined inside it. Each tenant gets its own organization, users and groups, RBAC scope, and API keys for automation. Tenant clusters register against the same CloudCasa server but cannot see each other's data. Destination backup storage locations (object and file storage) can be shared, with per-tenant buckets and credentials, or fully per-tenant. This is the deployment shape used by CSPs and MSPs bundling CloudCasa into a managed Kubernetes service.

DR Testing Considerations

Catalog Backup

CloudCasa's own state — clusters, policies, organizations, roles, schedules, job history — lives in MongoDB. The catalog is itself a critical asset. CloudCasa includes a built-in catalog backup CronJob that runs `casactl catalog backup` against the configured backup storage location. Recommended cadence is hourly. Recommended retention is at least 30 days, aligned with snapshot retention.

Catalog restore is straightforward: deploy a new CloudCasa server on the secondary management cluster (Helm or ArgoCD), then run `casactl` to restore the catalog from the backup storage location with the snapshot ID. The catalog comes back, agents reconnect, and the secondary becomes the new primary. Once Site A is recovered, the same mechanism is used to fail back.

[Catalog backup/restore - CloudCasa Self-hosted Admin Guide](#)

Backup of the Catalog: The Most Critical Job

The single most important policy in this architecture is the CloudCasa catalog backup itself. If the catalog is lost and there is no off-cluster catalog snapshot, the backups in the backup storage location are still recoverable (the data is intact) but cluster, policy, schedule, and RBAC state have to be rebuilt by hand before they can be addressed. Run catalog backups hourly, retain for at least 30 days, and rehearse catalog restore at least quarterly.

Disaster Recovery Strategy

DR for OCP-V workloads follows the same application-centric pattern as for containerized apps. CloudCasa does not aim to restore an OpenShift cluster itself: the recommended pattern is to keep a secondary OpenShift cluster available (either fully deployed, or quickly provisionable on Site B hardware) and to restore the workloads onto it from the off-cluster object store.

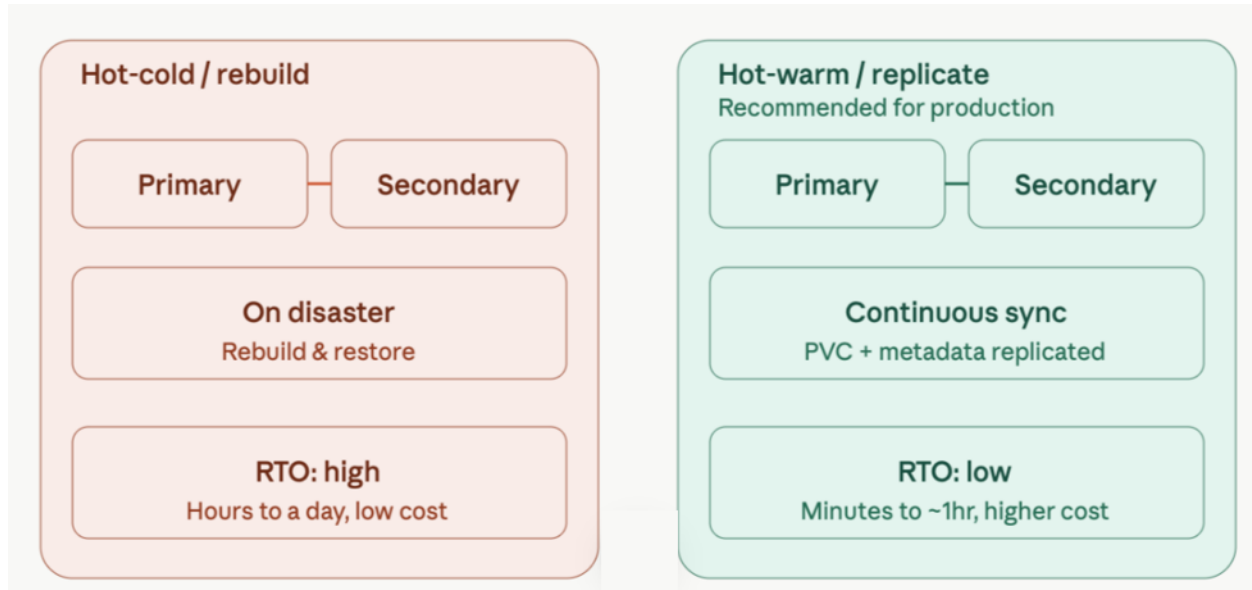


FIGURE 17 DR STRATEGIES — REBUILD (HOT-COLD) VS REPLICATE (HOT-WARM), BOTH ON-PREMISES

Strategy	RTO	Infrastructure cost	Use case
Rebuild (hot-cold)	High (hours to a day)	Low (no idle Site B cluster)	Pre-production, small organizations, constrained budgets
Replicate (hot-warm)	Low to medium (minutes to ~1 hour)	Higher (warm Site B cluster)	Production and business-critical workloads
Replicate + CloudCasa DR	Lowest (seconds, minutes) (RPO RTO)	Highest	Mission-critical workloads with strict RPO requirements

* CloudCasa DR is an optional, paid feature that uses storage-level asynchronous / synchronous replication on supported storage platforms. It is not required for the architecture described here but is available when RPO targets are tighter than backup-export cadence allows.

RTO and RPO

RPO is determined before a disaster — the maximum amount of unrecoverable transaction time prior to a failure. RTO, while also determined before a disaster, is measured after the event —

the time to make services available for new transactions. Both are time measures, but they measure different things.

- RPO is limited by the backup frequency or snapshot cadence. If using synchronous storage-based replication with CloudCasa DR, it is possible to meet RPOs of near zero.
- RTO is bounded by the time it takes to restore selected workloads (a function of data size and restore throughput or failover speed for CloudCasa DR).

DR Testing

A backup is only useful if it can be restored. The architecture supports recurring, non-disruptive DR testing because restores can target an alternate namespace, an alternate storage class, or an alternate cluster. Common patterns:

- DR Failover and Failback: Create a DR plan using supported storage arrays and enable clusters connected to that array for DR. Create failover and failback recovery plans within the DR tab apply them to Recovery jobs where applications, namespaces, VMs, Deployments, StatefulSets, etc. can be failed over and failed back. Perform the same operation using CloudCasa server DR failover / failback.
- Sandbox restore: restore a production namespace into a sandbox namespace on the same cluster, validate, then delete.
- Site B drill: scheduled monthly restore of selected applications onto Site B; measure RTO and verify application functionality.
- Catalog restore drill: rebuild a CloudCasa server from catalog backups onto a third cluster to confirm the catalog itself is recoverable.

Performance Tuning

Memory Utilization / Configuration Recommendations

Agent Resource Footprint

CloudCasa's agent runs as several containers under an Agent Manager, which handles deploying and updating agent components on each cluster, and ships with default CPU and memory requests/limits for these containers. Rather than being fixed, these defaults can be overridden directly in the CloudCasa UI: under Add/Edit Cluster → Advanced options → Customize agent installation parameters, administrators can supply a YAML document that sets resource requests, limits, and ephemeral storage for individual containers (kubeagent, kubeagent-backup-helper,

kubeagentmanager, data-mover, and azure-files-mover), along with labels, annotations, tolerations, and security context at the pod, container, or namespace level.

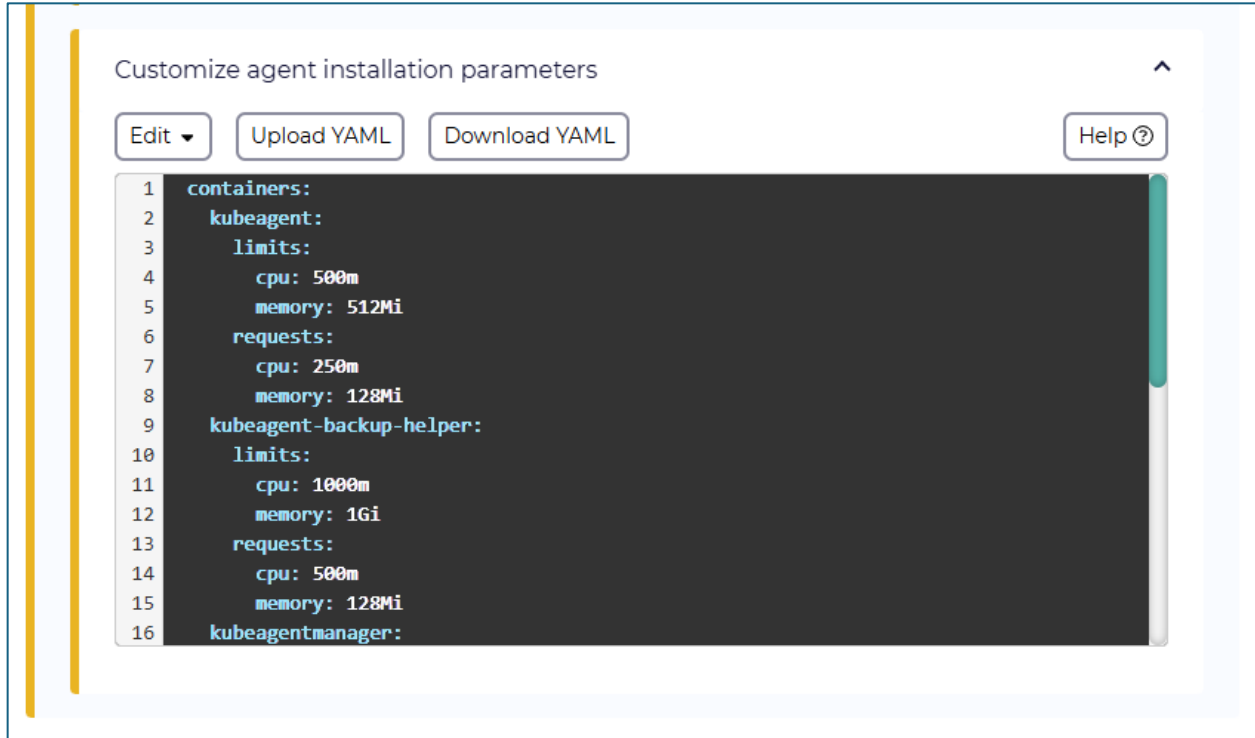


FIGURE 18 CUSTOMIZE AGENT INSTALLATION PARAMETERS SERVER RESOURCE FOOTPRINT

Resource requirements depend on number of clusters, number of jobs, churn rate, and data type. The defaults below are documented to support at least 100 clusters, 100 policies, and 100 concurrent jobs without tuning¹.

Component	CPU req / limit (m)	Memory req / limit	Replicas
apiserver	250 / 500	512 Mi / 1 Gi	3
kubeagentserver (kas)	100 / 250	256 Mi / 1 Gi	2
eventreconciler	50 / 250	512 Mi / 2 Gi	3

¹ <https://docs.cloudcasa.io/self-hosted-admin-guide/resourcelimits.html>

gandalf	25 / 50	128 Mi / 1 Gi	1
amds-frontend	10 / 25	16 Mi / 64 Mi	1
amds-envoy	10 / 25	16 Mi / 64 Mi	1
grpcapiserver	10 / 25	16 Mi / 64 Mi	1
Dex	5 / 25	32 Mi / 64 Mi	1
Redis	10 / 25	32 Mi / 64 Mi	1
Minio	25 / 50	256 Mi / 512 Mi	1
MongoDB (default install)	25 / 50	512 Mi / 1 Gi	1

Appendix: Additional Operational and Security Considerations

Authentication and Authorization

CloudCasa Self-Hosted federates authentication through Dex. Supported upstream providers are LDAP/LDAPS, generic OIDC (including Keycloak and Okta), AWS Cognito, Google OAuth 2.0, and Azure OAuth2. A local provider is supported for proof-of-concept only. Site-admin permissions are granted via the `auth.siteadminUsers` and `auth.siteadminUserGroups` Helm values.

Authorization inside CloudCasa is role-based. Operators define roles with explicit permissions on the data plane and on platform configuration, then map roles to user groups. External groups from the auth provider are matched by name to CloudCasa user groups, so a user added to the right LDAP group inherits the right CloudCasa role automatically. Combined with multi-tenant organizations, this is sufficient for MSP-style separation of tenants on a single CloudCasa instance.

For OpenShift specifically, deploy with `scc.enabled` set to `true` (and `scc.priority` set as needed) so CloudCasa creates its own `SecurityContextConstraint` resources rather than depending on cluster-admin role grants for its service accounts.

GitOps Lifecycle with Red Hat OpenShift GitOps (ArgoCD)

The CloudCasa server Helm chart is ArgoCD-friendly. Deploy via Red Hat OpenShift GitOps so the server, MongoDB, secrets, certificates, and any pre-install resources are all driven from a Git repository. The recommended pattern is:

- Static manifests for namespace, image pull secret, cert-manager Issuer, and any RBAC required pre-install. Marked with `argocd.argoproj.io/sync-wave: "-1"`.
- Helm chart for the CloudCasa server itself. Pull either from `helm.cloudcasa.io` with supplied credentials or from a mirrored internal Helm repository for air-gapped sites.
- Custom values file in Git, with secrets sourced via External Secrets Operator or SealedSecrets — never committed in clear text.
- Set `mongo.createNamespace: false` so ArgoCD owns the `cloudcasa-mongo` namespace lifecycle instead of Helm.

ArgoCD then gives you the same standardized, repeatable deployment for both the primary and secondary management clusters, and any future cluster additions. It also makes upgrades a Git operation.

Provisioning CloudCasa with Terraform

The CloudCasa Terraform Provider lets you manage CloudCasa resources as infrastructure-as-code, using Terraform to configure clusters, backup jobs, policies, and storage locations, and to automatically deploy the CloudCasa agent to OpenShift clusters.

The provider exposes several resource types. A `cloudcasa_kubecoluster` represents an OpenShift cluster and can auto-install the CloudCasa agent using the local `kubeconfig` context. A `CloudCasa_objectstore` defines S3 or Azure Blob backup storage locations, which can be attached to a cluster or to individual backup jobs. A `CloudCasa_kubebackup` resource defines snapshot or copy backup jobs, either ad hoc (via `run_on_apply`) or scheduled, and `cloudcasa_policy` resources define retention and Cron-based backup schedules.

A common use case is wiring cluster provisioning and backup protection together in one pipeline, so a new cluster automatically gets the agent installed and backup policies applied as part of `terraform apply`. Existing CloudCasa resources created via the UI can also be brought under

Terraform management using terraform import or Terraform 1.4+ import blocks, though once imported, local Terraform config takes precedence over changes made in the CloudCasa UI. The provider itself is open source under Apache 2.0, maintained by Catalogic Software on GitHub.

More information can be found here:

<https://registry.terraform.io/providers/catalogicsoftware/cloudcasa/latest>

Hardening with Red Hat Advanced Cluster Security (RHACS)

CloudCasa needs broad read and write permissions inside the protected clusters and on the backup target. It should be monitored and protected as a critical component of the platform. RHACS (StackRox) is the right tool for that on OpenShift:

- Restrict secret access in the cloudcasa-server and cloudcasa-mongo namespaces. Alert on any GET, UPDATE, DELETE, or PATCH on those namespaces' secrets where the actor is not the CloudCasa service account.
- Alert on Kubernetes Actions: Exec into pod for any pod in cloudcasa-server, cloudcasa-mongo, or cloudcasa-io. There are very few legitimate cases for exec into a backup pod; treat them as anomalies that need an approver.
- Apply an admission control policy that prevents the deployment of unsigned CloudCasa container images. Combined with image signing (cosign / sigstore) at the registry, this defends against tampered backup binaries.
- Audit-log any change to the CloudCasa Helm release or to the ArgoCD Application that manages it.

Air-Gapped Installation

Both Helm-based and ArgoCD-based installations work in air-gapped environments. The flow is:

1. Download the CloudCasa server Helm chart .tgz and the full image list from the CloudCasa support portal.
2. Mirror all images into your internal registry under a single root path (the helm value images.registry must point at that path; images.agentRegistry must be set to null to disable the default public pull).
3. Create the cloudcasa-server namespace, image pull secret, and cert-manager Issuer.
4. Run helm install (or sync the ArgoCD application) using the local .tgz file.
5. Once amds-envoy is reachable externally, set amdsEnvoyUrl to the external address and run helm upgrade (or sync) to refresh TLS certificates and ingress configuration.

The CloudCasa agent images live in a separately, publicly accessible registry by default. For fully air-gapped client clusters, mirror those too and set the agent images.registry value in the cluster's agent configuration.

Network Configuration

Network

OpenShift Virtualization on the workload cluster uses the OVN-Kubernetes default network for pod-to-pod traffic. VM workloads that need direct access to physical networks (for example, legacy VLAN-attached VMs) get additional networks via the NMState operator, a Linux bridge, and Multus. For this reference architecture, the workload cluster is configured with:

- Default OVN-Kubernetes overlay for east-west pod traffic.
- Linux bridge networks via NMState for any VMs that need flat layer-2 access.
- OpenShift Routes for north-south application traffic, including the CloudCasa server's amds-envoy ingress.

The CloudCasa server's amds-envoy endpoint must be reachable from every workload cluster's agent on TCP 443. In a typical on-premises deployment this is an internal DNS name that resolves to a LoadBalancer service (MetalLB or a cluster ingress LoadBalancer) or to an OpenShift Route.

CloudCasa Configuration

Helm Values Overview

The following Helm values matter the most for this architecture:

Value	Purpose
storageClassName	Block-backed StorageClass for the MongoDB PVC and the log bucket.
licenseKey	CloudCasa license key bound to the management cluster's kube-system UID.
serviceType	Set to LoadBalancer (with MetalLB or a hardware LB) or use the OpenShift Route. Default is NodePort.

certificates.issuerName	cert-manager Issuer name. Use ClusterIssuer for cluster-wide cert lifecycle.
auth.allowedUsers, auth.allowedUserGroups	Users and groups (from the upstream auth provider) permitted to log in.
auth.siteadminUsers, auth.siteadminUserGroups	Site administrators with full platform access.
scc.enabled	Set to true on OpenShift 4.14+. Creates CloudCasa-specific SecurityContextConstraints.
mongo.createStatefulset	Set to false and provide mongo.url + mongo.authSecretName for an external MongoDB.
mongo.pv.size	Catalog PVC size. Default 16 Gi is sized for hundreds of clusters.
emails.*	SMTP or Azure Communication Service settings for notifications.
amdsEnvoyUrl	External hostname or IP of the CloudCasa server. Set after first install once the LB / Route is provisioned.

Site B Import / Restore

Site B does not require recurring import policies in CloudCasa's model because the secondary CloudCasa server reads the same backup destination-storage when activated. On a DR event:

1. Restore the CloudCasa catalog onto the Site B management cluster via `casactl`.
2. Re-register the Site B workload cluster (or, if the workload cluster was already registered, confirm the agent reconnects).
3. Trigger a Restore Action against the latest snapshot for each workload — namespace, label, or full policy scope.
4. Optional: use a transform set to remap StorageClass if Site B uses a different CSI driver.

Day-2 Operations Checklist

- Catalog backup runs hourly to the off-cluster backup storage location. Monitor the catalogbackup CronJob.

- Repository maintenance (expired backup pruning) runs daily per backup destination store. Monitor cloud task pod completion.
- Agent auto-update is on by default and runs every 24 hours. Track agent versions across the fleet from the CloudCasa UI.
- CloudCasa server upgrades follow the standard Helm or ArgoCD upgrade path.
- Certificate renewal: cert-manager handles the cluster-internal certificates. Routes that terminate TLS at the OpenShift ingress controller use whatever the cluster's ingress operator is configured for.

Monitoring

CloudCasa exposes Prometheus-scrappable metrics for server components. Recommended dashboards and alerts:

- Job success rate by cluster and by policy (target: > 99% over 7 days).
- Time-since-last-successful-backup per protected namespace (alert if greater than RPO).
- Catalog backup success (alert if the latest successful catalog backup is more than 4 hours old).
- Destination storage usage growth and pruning effectiveness.
- Agent connection state per cluster (alert if Pending for more than 10 minutes).

On OpenShift, the Prometheus stack from OpenShift Monitoring or User Workload Monitoring is the natural target. A reference Grafana dashboard ships with the CloudCasa server documentation.

References

ID	Reference
R1	Red Hat OpenShift Virtualization documentation — https://docs.RedHat.com/en/documentation/openshift_container_platform
R2	CloudCasa Self-Hosted Admin Guide — https://docs.cloudcasa.io/self-hosted-admin-guide/
R3	CloudCasa Self-Hosted datasheet (April 2026) — https://cloudcasa.io/assets/CloudCasa-SelfHosted-Datasheet-Apr-2026.pdf
R4	CloudCasa for Red Hat OpenShift — https://cloudcasa.io/partners/red-hat-openshift/
R5	CloudCasa user documentation — https://docs.cloudcasa.io/help/
R6	Red Hat Advanced Cluster Security for Kubernetes documentation
R7	Red Hat OpenShift GitOps (ArgoCD) documentation
R8	OpenShift Virtualization etcd backup script (CloudCasa) — https://github.com/catalogicsoftware/cloudcasa-artifacts

© 2026 Catalogic Software. All rights reserved. CloudCasa is a trademark of Catalogic Software, Inc.