EDB POSTGRES AI + Red Hat Ansible Automation Platform

VALIDATED REFERENCE ARCHITECTURE

Enterprise Automation Resilience

High availability and disaster recovery for Red Hat Ansible Automation Platform on EDB Postgres® AI

enterprisedb.com

Contents

Executive overview

In today's IT landscape, the central challenge is no longer whether to automate; it is whether your automation platform itself is resilient enough to trust with mission-critical workloads. A platform outage that silences your automation engine interrupts IT operations and also freezes every pipeline, deployment, and remediation that depends on it.

This paper describes a validated reference architecture for deploying Red Hat Ansible Automation Platform (AAP) 2.6 (Containerized) with backing from EDB Postgres AI (EDB PG AI), with high availability within a single data center. The architecture can be extended to an active/passive multi–data center configuration for disaster recovery (covered in Section 7). The result is an automation backbone that achieves sub-minute failover within a data center and sub–five-minute recovery for cross–data center disaster recovery (DR).

What this architecture delivers

Capability	Description
Automated high availability	EDB PG AI monitors cluster health, manages virtual IP, and handles automatic failover to maintain continuous database availability. Restore service in less than 60 seconds with near-zero data loss.
Simplified deployment	Familiar Ansible tools provision the entire database cluster, including the OS, PostgreSQL, and failover management, based on predefined, validated reference architectures.
Validated by Red Hat and EDB	Running AAP against an external EDB cluster immediately adds high availability (HA) capabilities to the database layer, following the enterprise standard for AAP deployment.

Why this matters

Organizations that have standardized on Ansible Automation Platform as their enterprise-wide automation strategy have made the database layer used with AAP foundational to every team that depends on automation. A resilient database is a critical safety net that keeps your entire automation strategy operational when failures occur.

Red Hat and EDB have jointly validated this configuration to deliver production-ready replication, automated failover, and backup integrity.

Key design summary

Aspect	Specification
Deployment method	AAP 2.6 Containerized Installer (Podman on RHEL 9.4+)
Topology	High availability within single data center
AAP nodes	8 dedicated component VMs
Database	3-node PostgreSQL cluster (1 primary + 2 standbys)
Replication	Physical streaming + WAL archiving
Database connectivity	EDB Failover Manager-managed virtual IP (VM/bare metal) or load balancer (cloud)
Automated failover	< 60 seconds RTO via Failover Manager orchestration

Note: This architecture is based on Red Hat's tested container enterprise topology. Section 7 extends this foundation to multi–data center active/passive DR.

1. Architecture overview

The architecture implements AAP 2.6 using the containerized installer (Podman on RHEL 9.4+) running on nodes within a single data center. EDB PG AI provides the externalized database layer, managed by Failover Manager for automated promotion and virtual IP management.

1.1. High-level architecture diagram

1.2. AAP platform components

Platform Gateway (2 nodes)

The Platform Gateway serves as the unified entry point for all AAP services. It performs reverse-proxy routing to backend components, renders the unified web UI, and handles single sign-on (SSO) authentication. Users interact with a single URL (e.g., `https://aap.example.com`) regardless of which backend service processes their request.

Automation Controller (2 nodes)

The Automation Controller is the scheduling and execution engine for Ansible automation. It provides the REST API and web interface for managing inventories, credentials, job templates, and workflows. Controller nodes are stateless—all job state is persisted in PostgreSQL—so if one controller fails mid-job, another can detect the orphaned job via database heartbeat and re-queue it. The task container runs Celery workers that dispatch jobs to execution environments, while the receptor container provides mesh networking to remote execution nodes.

Automation Hub (2 nodes)

The Automation Hub manages Ansible content: certified and custom collections, execution environment container images, and role-based access control for content namespaces. Both Hub instances share a persistent NFS volume mounted at `/var/lib/pulp` for binary storage—when one node receives an upload, the other can immediately serve downloads. Each Hub node runs a colocated Redis instance for caching and async task queuing.

Event-Driven Ansible (EDA) (2 nodes)

The EDA Controller enables event-driven automation by subscribing to event sources (Kafka, webhooks, Prometheus alerts, cloud events) and matching incoming events against Ansible Rulebooks. When conditions match, EDA triggers job templates and workflow templates on the Automation Controller via API. Each EDA node runs a colocated Redis instance for event state tracking and job queue management. Rulebook activation state is persisted to PostgreSQL, so activations survive node restarts.

1.3. Database components

EDB PG AI primary (1 node)

The primary database handles all read/write operations for AAP. It hosts four databases: `awx` (Controller), `automationhub` (Hub), `automationedacontroller` (EDA), and `automationgateway` (Gateway). The primary maintains write-ahead log (WAL) records that are streamed to standbys for replication. AAP components connect to the primary through the virtual IP, never directly, ensuring seamless failover.

EDB PG AI standbys (2 nodes)

Standby databases continuously replay WAL records from the primary via streaming replication. They operate in hot standby mode, capable of serving read-only queries for reporting or analytics workloads. The first standby (pg-dc1-2) uses synchronous replication for zero data loss; the second (pg-dc1-3) uses asynchronous replication to reduce primary latency. During failover, the standby with the least replication lag is promoted to primary.

EDB Failover Manager agents (3 instances)

Failover Manager agents run on each database node, forming a cluster that monitors health and orchestrates failover. Agents communicate via peer-to-peer protocol (port 7800), exchanging heartbeats every few seconds. When the primary becomes unreachable, agents vote on whether to promote a standby. With three agents, quorum requires two votes. This prevents split-brain scenarios in which network partitions could cause two nodes to both claim primary status. Failover Manager also manages the virtual IP, binding it to whichever node is currently primary.

Virtual IP (1 address)

The virtual IP (VIP) provides a stable database endpoint that AAP components use for all connections. The VIP is an IP address (e.g., `10.1.2.100`) that Failover Manager binds to the current primary's network interface. On failover, Failover Manager releases the VIP from the failed primary and binds it to the newly promoted primary, then sends a gratuitous ARP to update network switches. AAP connections fail briefly, then automatically reconnect to the same VIP—now pointing to the new primary—without any configuration change.

1.4. Component summary

Layer	Component	Count	Purpose
AAP	Platform Gateway	2	Reverse proxy, SSO, unified UI
AAP	Automation Controller	2	Job scheduling, API, execution
AAP	Automation Hub	2	Collections, execution environments
AAP	Event-Driven Ansible	2	Event processing, rulebooks
Database	PostgreSQL primary	1	Read/write operations
Database	PostgreSQL standby	2	Replication, failover candidates
Database	Failover Manager agent	3	Health monitoring, failover orchestration
Database	Virtual IP	1	Stable database endpoint

1.5. Data flow

```
User Request Flow:

User → Load Balancer (GLB) → HA Proxy → Platform Gateway → Controller/Hub/EDA →
Database VIP → PostgreSQL Primary

Replication Flow:

PostgreSQL Primary → Streaming Replication → Standby 1 (sync)
                   → Streaming Replication → Standby 2 (async)
                   → WAL Archive → S3/NFS
```

2. EDB Postgres layer

2.1. Cluster specifications

Key design summary

Role	Count	vCPU	Memory	Disk	Purpose
Primary	1	8	32 GB	500 GB SSD	Read/write operations
Standby	2	8	32 GB	500 GB SSD	Replication, failover
Total	**3**	**24**	**96 GB**	**1.5 TB**	

Database Layout
AAP requires four databases:

Database	Owner	Purpose	Estimated Size
awx	aap	Automation Controller	50 GB
automationhub	aap	Collections, images	20 GB
automationedacontroller	aap	Event-driven automation	10 GB
automationgateway	aap	Platform Gateway	5 GB

2.2. Network configuration

Database subnet

```
Database Subnet: 10.1.2.0/24
Hostnames and IPs:
  pg-dc1-1.example.com (Primary)    10.1.2.21
  pg-dc1-2.example.com (Standby)    10.1.2.22
  pg-dc1-3.example.com (Standby)    10.1.2.23
Virtual IP (EFM-managed):       10.1.2.100
WAN Connectivity:
  - Type: Site-to-Site VPN or Direct Connect
  - Bandwidth: 100 Mbps minimum, 1 Gbps recommended
  - Latency: < 100ms required for streaming replication
  - Encryption: IPsec or TLS
```

Firewall rules
(AAP is on 10.1.1.0/24)

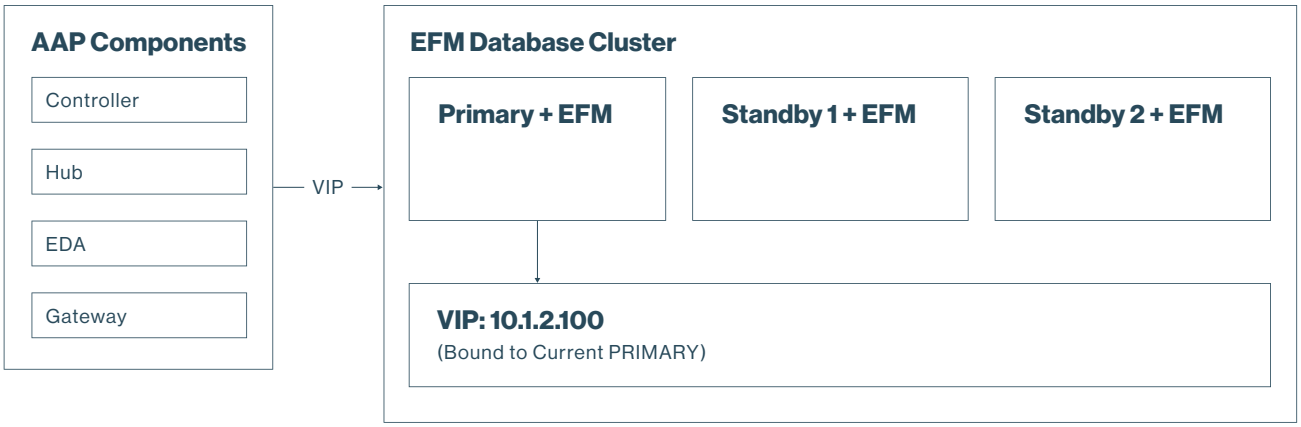
Traffic	Source	Destination	Port	Protocol
AAP → PostgreSQL	10.1.1.0/24	10.1.2.100 (VIP)	5432	TCP
Streaming replication	10.1.2.21-23	10.1.2.21-23	5432	TCP
Failover Manager cluster	10.1.2.21-23	10.1.2.21-23	7800	TCP
Failover Manager admin	Management	10.1.2.21-23	7809	TCP

2.3. Connectivity options

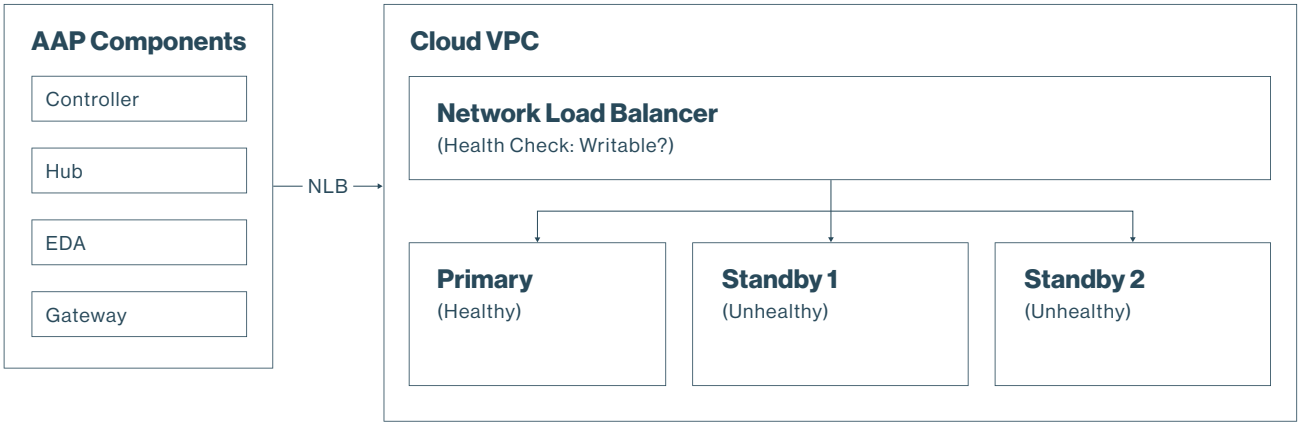
Red Hat AAP components need a stable endpoint to connect to the PostgreSQL primary. Since the primary can change during failover, a mechanism is required to route connections to the current primary.

Important: AAP manages database connections internally and is *not compatible* with external connection poolers such as PgBouncer. Connect AAP directly to the VIP or load balancer.

Option A: Failover Manager–managed VIP (VM/bare metal)—recommended



Option B: Cloud load balancer (AWS/Azure/GCP)
In cloud environments, use a TCP load balancer with health checks:



AAP inventory for cloud:

```
controller_pg_host='aap-db-nlb.us-east-1.elb.amazonaws.com'
```

2.4. Configuration files

postgresql.conf: See recommended defaults.

```
# /var/lib/pgsql/17/data/postgresql.conf
# Keep the defaults and validate the configuration settings below..

# --- Connection Settings ---
max_connections = 1500

# --- Memory ---
shared_buffers = 8GB
effective_cache_size = 24GB
work_mem = 64MB
maintenance_work_mem = 2GB

# --- Replication ---
wal_level = replica
max_wal_senders = 10
max_replication_slots = 10
synchronous_standby_names = 'pg-dc1-2' # Set on pg-dc1-1 only
synchronous_commit = on
wal_keep_size = 1GB
hot_standby = on
hot_standby_feedback = on

# --- Checkpoints ---
checkpoint_timeout = 15min
checkpoint_completion_target = 0.9

# --- Performance ---
random_page_cost = 1.1
effective_io_concurrency = 200
```

On the standby database on pg-dc-1-2, the `primary_conninfo` setting should include `application_name='pg-dc-2'`.

pg_hba.conf (primary): See recommended defaults.

```
# /var/lib/pgsql/17/data/pg_hba.conf
# TYPE  DATABASE               USER      ADDRESS             METHOD

# Local connections
local   all                    all                           peer
host    all                    all       127.0.0.1/32        scram-sha-256

# AAP application connections (from AAP subnet)
host    awx                     aap      10.1.1.0/24         scram-sha-256
host    automationhub           aap      10.1.1.0/24         scram-sha-256
host    automationedacontroller aap      10.1.1.0/24         scram-sha-256
host    automationgateway       aap      10.1.1.0/24         scram-sha-256

# Replication connections (from standbys)
host    replication            replicator 10.1.2.22/32       scram-sha-256
host    replication            replicator 10.1.2.23/32       scram-sha-256

# EFM health checks
host    postgres               efm        10.1.2.0/24        scram-sha-256
```

2.5. Failover Manager configuration

See recommended default efm.properties in Failover Manager.

```
# /etc/edb/efm-5.3/efm.properties
# --- Database Connection ---
db.user=efm
db.password.encrypted=<run: efm encrypt efm_password>
db.port=5432
db.database=postgres
db.service.owner=enterprisedb
db.service.name=edb-as-17
db.bin=/usr/edb/as17/bin
db.data.dir=/var/lib/pgsql/17/data

# --- Cluster Membership ---
bind.address=10.1.2.21:7800
# Example: bind.address=10.1.2.21:7800

# --- Admin ---
admin.port=7809
is.witness=false
promotable=true

# --- Failover ---
auto.failover=true
auto.reconfigure=true
promotable=true

# --- Timeouts - Adjust as per your requirements ---
local.timeout=60
local.period=10
remote.timeout=10
node.timeout=50

# --- VIP Configuration (VM/Bare Metal) ---
virtual.ip=10.1.2.100
virtual.ip.interface=eth0
virtual.ip.prefix=24
virtual.ip.single=true

# For cloud deployments (disable VIP, use script):
# virtual.ip=
# script.post.promotion=/usr/local/bin/update-cloud-lb.sh
```

2.6. Installation steps

Step 1: Install PostgreSQL (all nodes)
Please follow EDB Postgres AI installation procedures, and configure Failover Manager as per the documentation.

Step 2: Set up schema on the primary (pg-dc1-1)

```
# Once your database is running, setup the AAP databases

# Create AAP databases (from Section 3.1)
sudo -u postgres psql <<EOF
CREATE USER aap WITH ENCRYPTED PASSWORD '<aap_db_password>';
CREATE DATABASE awx OWNER aap;
CREATE DATABASE automationhub OWNER aap;
CREATE DATABASE automationedacontroller OWNER aap;
CREATE DATABASE automationgateway OWNER aap;
\c automationhub
CREATE EXTENSION IF NOT EXISTS hstore;
EOF
```

3. AAP platform layer

3.1. Component specifications

VM sizing

Component	Count	vCPU	Memory	Disk	Colocated services
Platform Gateway	2	2	8 GB	60 GB	Redis
Automation Controller	2	4	16 GB	60 GB	-
Automation Hub	2	4	16 GB	60 GB	Redis
Event-Driven Ansible	2	4	16 GB	60 GB	Redis
Total	**8**	**32**	**128 GB**	**480 GB**	

Containers per component

Platform gateway:

Container	CPU	Memory	Purpose
automation-gateway	1 core	2 GB	API gateway, routing
redis	1 core	4 GB	Session storage

Automation Controller:

Container	CPU	Memory	Purpose
automation-controller-web	2 core	8 GB	UI and API
automation-controller-task	1 core	4 GB	Job execution
automation-controller-web	1 core	2 GB	Mesh networking

Automation Hub:

Container	CPU	Memory	Purpose
automation-hub	2 core	8 GB	Content management
redis	1 core	4 GB	Cache

Event-Driven Ansible:

Container	CPU	Memory	Purpose
eda-activation-worker	2 core	8 GB	Event processing
redis	1 core	4 GB	Job queue

3.2. Network configuration

AAP subnet

```
AAP Subnet: 10.1.1.0/24

Hostnames and IPs:
  gateway1-dc1.example.com      10.1.1.11
  gateway2-dc1.example.com      10.1.1.12
  controller1-dc1.example.com   10.1.1.13
  controller2-dc1.example.com   10.1.1.14
  hub1-dc1.example.com          10.1.1.15
  hub2-dc1.example.com          10.1.1.16
  eda1-dc1.example.com          10.1.1.17
  eda2-dc1.example.com          10.1.1.18

WAN Connectivity:
  - Type: Site-to-Site VPN or Direct Connect
  - Bandwidth: 100 Mbps minimum, 1 Gbps recommended
  - Latency: < 100ms required for streaming replication
  - Encryption: IPsec or TLS
```

Firewall rules

Traffic	Source	Destination	Port	Protocol
User / GLB → HAProxy	0.0.0.0/0	10.1.1.100	443	TCP
HAProxy → Gateway	10.1.1.10	10.1.1.11-12	80/443	TCP
Gateway → Controller	10.1.1.11-12	10.1.1.13-14	8080/8443	TCP
Gateway → Hub	10.1.1.11-12	10.1.1.15-16	8081/8444	TCP
Gateway → EDA	10.1.1.11-12	10.1.1.17-18	8082/8445	TCP
AAP → database	10.1.1.0/24	10.1.2.100 (VIP)	5432	TCP
Redis cluster (if Redis HA)	10.1.1.11-12, 10.1.1.15-18	10.1.1.11-12, 10.1.1.15-18	6379, 16379	TCP
Controller → execution	10.1.1.13-14	<Exec nodes>	27199	TCP

3.3. Inventory file

The AAP inventory file defines where components are deployed and how they connect to the database.

```
# /opt/aap/inventory
# Red Hat Ansible Automation Platform 2.6 - Container Enterprise Topology

# ============================================================================
# AAP COMPONENT HOSTS
# ============================================================================

[automationgateway]
gateway1-dc1.example.com
gateway2-dc1.example.com

[automationcontroller]
controller1-dc1.example.com
controller2-dc1.example.com

[automationhub]
hub1-dc1.example.com
hub2-dc1.example.com

[automationeda]
eda1-dc1.example.com
eda2-dc1.example.com

# Redis colocated on gateway, hub, and EDA nodes (6 total for HA)
[redis]
gateway1-dc1.example.com
gateway2-dc1.example.com
hub1-dc1.example.com
hub2-dc1.example.com
eda1-dc1.example.com
eda2-dc1.example.com

# ============================================================================
# VARIABLES
# ============================================================================

[all:vars]
# --- Red Hat Registry ---
registry_username='<your RHN username>'
registry_password='<your RHN password>'

# --- Redis ---
redis_mode='standalone' # Use 'cluster' for Redis HA (optional)

# --- PostgreSQL Admin (for DB setup) ---
postgresql_admin_username='postgres'
postgresql_admin_password='<postgres_admin_password>'
```

```
# ============================================================================
# DATABASE CONNECTION
# ============================================================================
# All AAP components connect to the same database endpoint.
# This should be the EFM-managed VIP (VM/bare metal) or Load Balancer (cloud).
# --- Platform Gateway Database ---
gateway_pg_host='10.1.2.100'
gateway_pg_port='5432'
gateway_pg_database='automationgateway'
gateway_pg_username='aap'
gateway_pg_password='<aap_db_password>'

# --- Automation Controller Database ---
controller_pg_host='10.1.2.100'
controller_pg_port='5432'
controller_pg_database='awx'

# --- Automation Hub Database ---
hub_pg_host='10.1.2.100'
controller_pg_username='aap'
controller_pg_password='<aap_db_password>'
hub_pg_port='5432'
hub_pg_database='automationhub'
hub_pg_username='aap'
hub_pg_password='<aap_db_password>'

# --- Event-Driven Ansible Database ---
eda_pg_host='10.1.2.100'
eda_pg_port='5432'
eda_pg_database='automationedacontroller'
eda_pg_username='aap'
eda_pg_password='<aap_db_password>'

# ============================================================================
# ADMIN CREDENTIALS
# ============================================================================

gateway_admin_password='<gateway_admin_password>'
controller_admin_password='<controller_admin_password>'
hub_admin_password='<hub_admin_password>'
eda_admin_password='<eda_admin_password>'

# --- Main URL (external access point) ---
gateway_main_url='https://aap.example.com'
```

Important: The `*_pg_host` values point to the database VIP (`10.1.2.100`). This VIP is managed by Failover Manager and automatically moves to the current primary. AAP does not need reconfiguration when database failover occurs.

3.4. Installation steps

Install AAP

```
# On installer node (can be gateway1):

# 1. Download and extract installer
cd /opt
tar -xzf ansible-automation-platform-containerized-setup-2.6-1.tar.gz
cd ansible-automation-platform-containerized-setup-2.6-1

# 2. Copy inventory file
cp inventory-dc1 ./inventory

# 3. Run installer
./setup.sh

# Installation takes 15-30 minutes
```

Enable services

```
# On each AAP node, enable the appropriate services:

# Gateway nodes:
systemctl enable --now automation-gateway
systemctl enable --now redis

# Controller nodes:
systemctl enable --now automation-controller-web
systemctl enable --now automation-controller-task

# Hub nodes:
systemctl enable --now automation-hub
systemctl enable --now redis

# EDA nodes:
systemctl enable --now eda-activation-worker
systemctl enable --now redis
```

Verify deployment

```
# Check containers are running
podman ps --format "table {{.Names}}\t{{.Status}}"

# Test API endpoints
curl -k https://gateway1.example.com/api/gateway/v1/health/
curl -k https://controller1.example.com/api/v2/ping/
curl -k https://hub1.example.com/api/galaxy/v3/
curl -k https://eda1.example.com/api/eda/v1/health/

# Verify database connectivity from each component
podman exec automation-controller-web \
  psql -h 10.1.2.100 -U aap -d awx -c "SELECT version();"
```

4. Failover procedures

4.1. Automated failover (via Failover Manager)

Trigger condition: PostgreSQL primary in DC1 becomes unavailable.

Automated failover sequence

```
1. EFM Detects Primary Failure (pg-dc1-1)
   - No other agent can also reach the database on pg-dc1-1
   - Decision: Initiate failover
   - Primary agent drops the VIP
2. EFM Promotes the most up to date standby
   - Command: pg_ctl promote
   - Standby becomes read-write primary
3. EFM Updates VIP
   - VIP is brought up on promoted standby
   - AAP connections redirect to new primary
```

Trigger condition: Primary node DC1-1 becomes unavailable.

Automated failover sequence

```
1. EFM Agents detect Primary (pg-dc1-1) has disappeared
   - No agent can also reach the database on pg-dc1-1
   - Decision: Initiate failover
   - Check to make sure the VIP is not available before continuing
2. EFM Promotes the most up to date standby
   - Command: pg_ctl promote
   - Standby becomes read-write primary
3. EFM Updates VIP
   - VIP is brought up on promoted standby
   - AAP connections redirect to new primary
```

5. Security considerations

5.1. Postgres TLS/SSL configuration

```
# postgresql.conf
ssl = on
ssl_cert_file = '/etc/pki/tls/certs/pg-server.crt'
ssl_key_file = '/etc/pki/tls/private/pg-server.key'
ssl_ca_file = '/etc/pki/tls/certs/ca-bundle.crt'
ssl_min_protocol_version = 'TLSv1.2'
```

5.2. Authentication

Component	Method
PostgreSQL users	SCRAM-SHA-256
Replication	SCRAM-SHA-256 with dedicated user
Failover Manager	Encrypted password in efm.properties
AAP	Credentials in inventory, stored encrypted

5.3. Secrets management

```
# Ansible Vault for passwords
ansible-vault encrypt_string 'ChangeMeDB123!' --name 'pg_password'

# PostgreSQL SCRAM-SHA-256
CREATE ROLE aap LOGIN PASSWORD 'SCRAM-SHA-256$...' ENCRYPTED;
```

6. Backups and disaster recovery of PostgreSQL servers

EDB Barman provides centralized backup and point-in-time recovery (PITR) for the EDB Postgres AI cluster supporting AAP. While Failover Manager handles automated failover for high availability, Barman handles the *durability* of your data, protecting against logical corruption, accidental deletes, ransomware, and the kinds of failure scenarios that prompt you to recover to a specific moment in time rather than just promote a standby.

A dedicated Barman server pulls base backups from the PostgreSQL primary using the streaming backup method, continuously receives write-ahead log (WAL) files via streaming replication, and stores them in a backup catalog with configurable retention. Barman is fully integrated with Failover Manager. When a failover occurs, Barman automatically re-points to the new primary without manual intervention.

6.1. Barman server specifications

Node sizing

Role	Count	vCPU	Memory	Disk	Purpose
Barman Server	1	4	16 GB	1 TB (separate volume for catalog)	Centralized backup and WAL archive

Storage sizing guidance: Plan for retention period × (full backup size + daily WAL volume). For the AAP database layout (~85 GB total) with 14-day retention and ~50 GB/day WAL volume, allocate ~1 TB. Adjust upward for longer retention, higher WAL generation, or compliance archives.

Backup catalog layout

Catalog component	Path	Purpose
Base backups	/var/lib/barman/aap-dc1/base/	Full base backups
WAL archive	/var/lib/barman/aap-dc1/wals/	Continuous WAL files
SSH config	/var/lib/barman/.ssh/	Streaming replication credentials
Barman config	/etc/barman.d/aap-dc1.conf	Per-server backup configuration

6.2. Backup strategy

Parameter	Setting	Rationale
Backup method	postgres (streaming)	Uses pg_basebackup over the wire; no SSH-to-postgres dance required for backup data
WAL method	streaming via replication slot	Continuous, low-latency WAL capture with no data loss risk on archive_command failure
Compression	gzip on WAL, optionally on base	Reduces storage footprint
Backup frequency	Daily full base backup	WAL streaming provides continuous incremental coverage
Retention policy	RECOVERY_WINDOW of 14 days	Ensures recoverability across two business weeks
Verification	barman check + barman verify-backup	Daily catalog integrity checks

6.3. Configuration files

Global Barman configuration: See EDB Barman docs.

```
# /etc/barman.conf

[barman]
barman_user = barman
configuration_files_directory = /etc/barman.d
barman_home = /var/lib/barman
log_file = /var/log/barman/barman.log
log_level = INFO
compression = gzip
retention_policy_mode = auto
retention_policy = RECOVERY WINDOW OF 14 DAYS
wal_retention_policy = main
minimum_redundancy = 2
```

Per-server backup configuration:

```
# /etc/barman.d/aap-dc1.conf

[aap-dc1]
description = "AAP PostgreSQL Cluster - DC1"
conninfo = host=10.1.2.100 user=barman dbname=postgres
backup_method = postgres
streaming_conninfo = host=10.1.2.100 user=streaming_barman
streaming_archiver = on
slot_name = barman_aap_dc1
create_slot = auto
backup_options = concurrent_backup
parallel_jobs = 4
network_compression = true

# Per-server retention overrides (optional)
retention_policy = RECOVERY WINDOW OF 14 DAYS
wal_retention_policy = main
```

Note: `conninfo` uses the Failover Manager–managed VIP (10.1.2.100). This means Barman automatically tracks the current primary, with no manual re-pointing required after a Failover Manager failover.

pg_hba.conf additions on primary

Add the following rules to the existing pg_hba.conf entries shown in Section 2.4:\`# Barman connections`

```
host  postgres     barman             10.1.2.30/32    scram-sha-256
host  replication  streaming_barman   10.1.2.30/32    scram-sha-256
```

6.4. Installation steps

Step 1: Install Barman on the backup server

Follow the EDB Barman installation procedures for your distribution. On RHEL 9:

```
# On barman-dc1
sudo dnf install barman barman-cli -y
```

Step 2: Create Barman database users on the primary

```
# On pg-dc1-1 (current primary)
sudo -u postgres psql <<EOF
CREATE USER barman WITH SUPERUSER ENCRYPTED PASSWORD '<barman_password>';
CREATE USER streaming_barman WITH REPLICATION ENCRYPTED PASSWORD
'<streaming_password>';
EOF
```

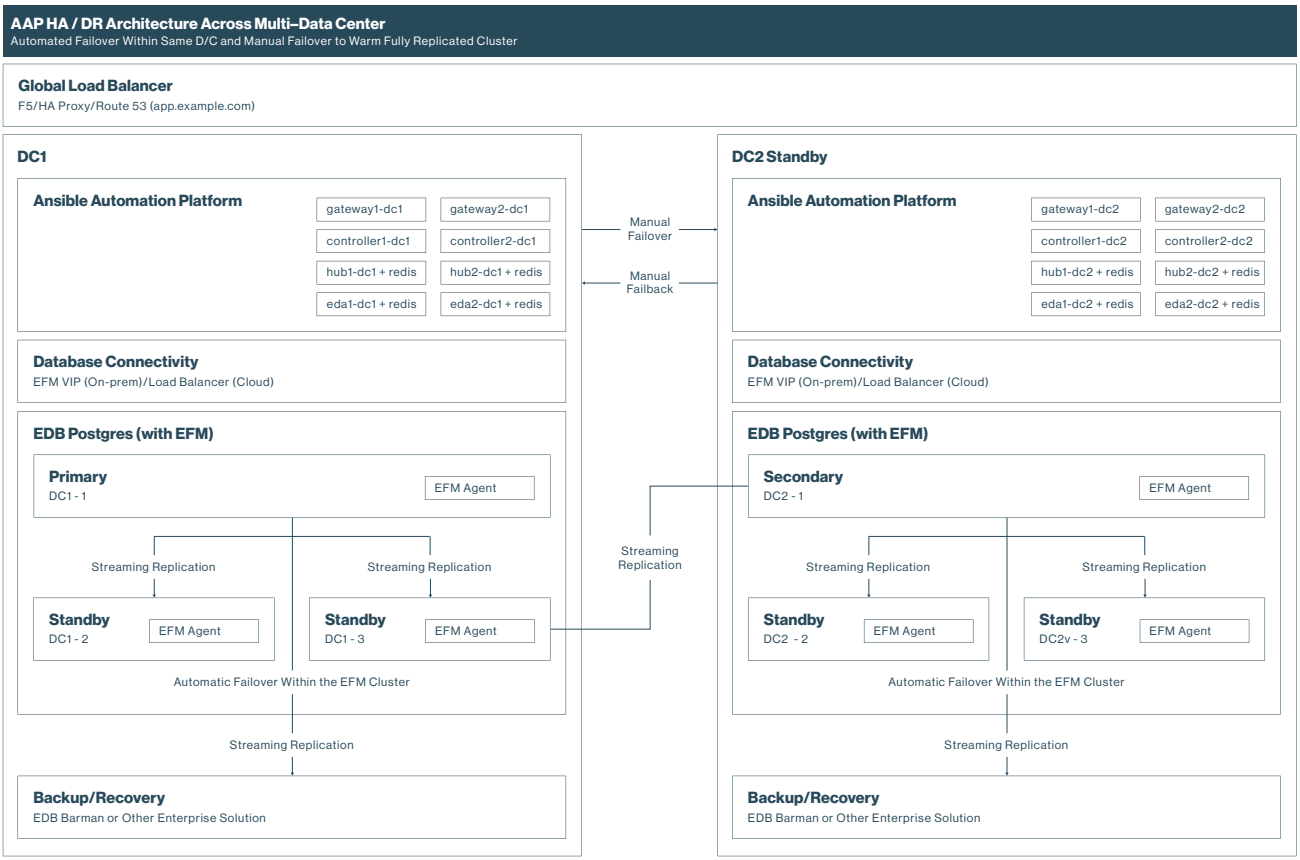
Note: The `barman` user requires SUPERUSER for backup metadata operations. Restrict access via pg_hba.conf and network rules accordingly.

Step 3: Store credentials in Barman's .pgpass

7. Multi–data center DR extension

This section extends the single–data center HA architecture to provide disaster recovery across two data centers.

7.1. DR architecture overview



7.2. AAP DC2 configuration

AAP in DC2 is installed but kept stopped until failover.

```
# DC2 inventory - points to DC2 database VIP
controller_pg_host='10.2.2.100'
hub_pg_host='10.2.2.100'
eda_pg_host='10.2.2.100'
gateway_pg_host='10.2.2.100'
```

```
# After DC2 installation, immediately stop all services:
systemctl stop automation-controller-web automation-controller-task
systemctl stop automation-gateway automation-hub eda-activation-worker redis
systemctl disable automation-*
```

7.3. EDB Postgres AI DC2 configuration

Cross-DC replication topology

```
DC1 PostgreSQL Cluster:
  pg-dc1-1 (PRIMARY)
   ├─> [other nodes ommitted]
   └─> pg-dc2-1 (DESIGNATED PRIMARY) - async standby (WAN)

DC2 PostgreSQL Cluster:
  pg-dc2-1 (DESIGNATED PRIMARY / STANDBY)
    ├─> pg-dc2-2 (STANDBY) - sync standby
    ├─> pg-dc2-3 (STANDBY) - async standby
    └─> S3/Barman (WAL Archive)
```

Replication configuration

Standby database creation:

```
# On pg-dc2-1 (designated primary for DC2)
pg_basebackup -h pg-dc1-1 -U replicator -D /var/lib/edb/as17/data \
  -P -Xs -R --slot=pg_dc2_1_slot -C

# On pg-dc2-2 (DC2 local standby)
pg_basebackup -h pg-dc2-1 -U replicator -D /var/lib/edb/as17/data \
  -P -Xs -R --slot=pg_dc2_2_slot

# On g-dc2-3 (DC2 local standby)
pg_basebackup -h pg-dc2-1 -U replicator -D /var/lib/edb/as17/data \
  -P -Xs -R --slot=pg_dc2_3_slot

# postgresql.auto.conf (auto-generated by -R flag)
primary_conninfo = 'host=pg-dc1-1 port=5432 user=replicator password=xxx
sslmode=verify-ca'
primary_slot_name = 'pg_dc2_1_slot'
recovery_target_timeline = 'latest'
```

Failover Manager configuration

The only unusual property required is a script called when a standby in the cluster is promoted. This can be used to start the AAP containers if a standby is being promoted to switch from one data center to another. The notification level is also changed to avoid INFO-level notifications.

```
# /etc/edb/efm-5.3/efm.properties

# Post-promotion Script (AAP integration)
script.post.promotion=/usr/edb/efm-5.3/bin/efm-orchestrated-failover.sh %h %s %a %v

# Notification
notification.level=WARNING
user.email=ops@example.com
```

Failover Manager integration script

```
#!/bin/bash
# /usr/edb/efm-5.3/bin/efm-orchestrated-failover.sh

set -e

CLUSTER_NAME="$1"
NODE_TYPE="$2"
NODE_ADDRESS="$3"
VIP_ADDRESS="$4"

# Determine datacenter
if [[ "$NODE_ADDRESS" == *"dc2"* ]] || [[ "$NODE_ADDRESS" == "10.2"* ]]; then
    DATACENTER="DC2"
    GATEWAY_NODES=("gateway1-dc2" "gateway2-dc2")
    CONTROLLER_NODES=("controller1-dc2" "controller2-dc2")
    HUB_NODES=("hub1-dc2" "hub2-dc2")
    EDA_NODES=("eda1-dc2" "eda2-dc2")
else
    echo "ERROR: Failover to DC1 not expected"
    exit 1
fi

# Start AAP containers by component type
echo "Starting Platform Gateway nodes in $DATACENTER..."
for node in "${GATEWAY_NODES[@]}"; do
    ssh "$node" "systemctl start automation-gateway redis"
done

echo "Starting Automation Controller nodes in $DATACENTER..."
for node in "${CONTROLLER_NODES[@]}"; do
    ssh "$node" "systemctl start automation-controller-web automation-controller-task"
done

echo "Starting Automation Hub nodes in $DATACENTER..."
for node in "${HUB_NODES[@]}"; do
    ssh "$node" "systemctl start automation-hub redis"
done

echo "Starting Event-Driven Ansible nodes in $DATACENTER..."
for node in "${EDA_NODES[@]}"; do
    ssh "$node" "systemctl start eda-activation-worker redis"
done

# Wait for AAP API
MAX_WAIT=300
ELAPSED=0
while [ $ELAPSED -lt $MAX_WAIT ]; do
    if curl -k -s https://10.2.1.100/api/v2/ping/ | grep -q "200"; then
        echo "AAP is ready in $DATACENTER"
        break
    fi
    sleep 10
    ELAPSED=$((ELAPSED + 10))
done

# Send notifications
logger -t efm-failover "AAP activated in $DATACENTER"
```

7.4. Manual switchover procedures between DC1 and DC2

To switch from DC1 to DC2:

```
# 1. Verify replication lag is acceptable
ssh pg-dc1-1 "psql -U postgres -c \"SELECT * FROM pg_stat_replication;\""

# 2. Stop AAP in DC1 (all component VMs)
for node in gateway1-dc1 gateway2-dc1; do
    ssh "$node" "systemctl stop automation-gateway redis"
done
for node in controller1-dc1 controller2-dc1; do
    ssh "$node" "systemctl stop automation-controller-web automation-controller-task"
done
for node in hub1-dc1 hub2-dc1; do
    ssh "$node" "systemctl stop automation-hub redis"
done
for node in eda1-dc1 eda2-dc1; do
    ssh "$node" "systemctl stop eda-activation-worker redis"
done

# 3. Shut down the database servers in DC1

# 4. Promote DC2 database to primary. The first step ensures dc2-1 has the highest
standby priority
ssh pg-dc2-1 "sudo /usr/edb/efm-5.3/bin/efm set-priority <cluster name> pg-dc2-1 1"
ssh pg-dc2-1 "sudo /usr/edb/efm-5.3/bin/efm force-promote <cluster name>"

# 5. Verify promotion
ssh pg-dc2-1 "psql -U postgres -c \"SELECT pg_is_in_recovery();\""

# Expected: f (false - not in recovery)

# 6. Update Global Load Balancer to DC2

# (Via GLB management interface)

# 7. Verify traffic flows to DC2
curl -k https://aap.example.com/api/v2/ping/
```

To switch back to DC1 after the above, please follow the steps to recreate DC1. To do so, create the database server on pg-dc1-1 as a standby following the primary in DC2. Then switch to DC1 following the above procedure.

**It is recommended that cross-data center failover is completed as controlled/manual switchover, not automated. Automated failover is recommended for single-data center setups.*

8. Building your automation foundation for the long term

Platform and infrastructure leaders are under increasing pressure to deliver automation at enterprise scale, while simultaneously guaranteeing that the systems their teams depend on will be available when it matters most. The architecture described in this brief is designed to meet both of those demands.

Red Hat Ansible Automation Platform 2.6 + EDB PG AI represents a jointly validated, production-ready foundation for organizations serious about automating data infrastructure with confidence. By externalizing the AAP database onto a dedicated, high-availability EDB cluster with automated failover, continuous WAL archiving, and event-driven monitoring, you move beyond basic uptime guarantees to a genuinely resilient automation backbone.

The three decisions that define long-term automation resilience:

1. **Externalize the database.** Running AAP with an internal, single-node database is a liability at enterprise scale. An externalized EDB cluster with Failover Manager gives the database layer its own independent HA and DR posture.
2. **Automate the failover.** Failover Manager–orchestrated failover removes humans from the critical path during a disaster scenario, converting a potentially 30-minute manual recovery into a sub-5-minute automated event.
3. **Close the loop with Event-Driven Ansible.** Using EDA to monitor and respond to platform health events means your automation platform protects itself continuously, without human polling.

The implementation roadmap in Section 7 is structured so that each phase delivers stand-alone operational value. Phase 1 alone—AAP 2.6 containerized with an external EDB cluster—represents a significant resilience improvement over internal-database deployments. Phases 2 through 4 progressively extend that foundation to full multi–data center DR with automated remediation.

For organizations standardizing on automation as a platform strategy, the question is not whether the automation platform needs a resilient database—it does. The question is whether that resilience is built in from the start or discovered under pressure. This architecture is designed to ensure that it's the former.

Related documentation

- EDB Ansible landing page
- EDB Postgres AI
- Solution Guide: High-Availability Ansible Automation Platform with EDB PostgreSQL Active-Passive DR
- EDB Ansible documentation
- Enterprise Automation Resilience with EDB and Red Hat Ansible Automation Platform
- The Sovereignty Gap: Why the Shift to SaaS-First Postgres Is Your Signal to Modernize with EDB and Red Hat
- Red Hat Ansible Automation Platform and EDB Postgres AI
- EDB Failover Manager: Managing Postgres database clusters and enabling high availability of primary-standby deployment architectures using streaming replication
- EDB Barman: Backup and Recovery Manager administration tool for remote backups and disaster recovery of PostgreSQL servers

External references

- Red Hat AAP 2.6 Container Growth Topology
- New Features and Enhancements: AAP 2.6 Containerized installation guide