AWS Migration Tool: KT Deep Dive

This presentation covers the technical architecture, features, and implementation details of our AWS Migration Tool. We'll explore how it streamlines AWS account migrations across organizations.

by Muhammed Abnas

Project Overview

Purpose & Architecture

A web application for AWS account migration between organizations.

- Full-stack application with React frontend
- FastAPI backend for robust API services
- Hosted on EC2 with Load Balancer

Current Deployment

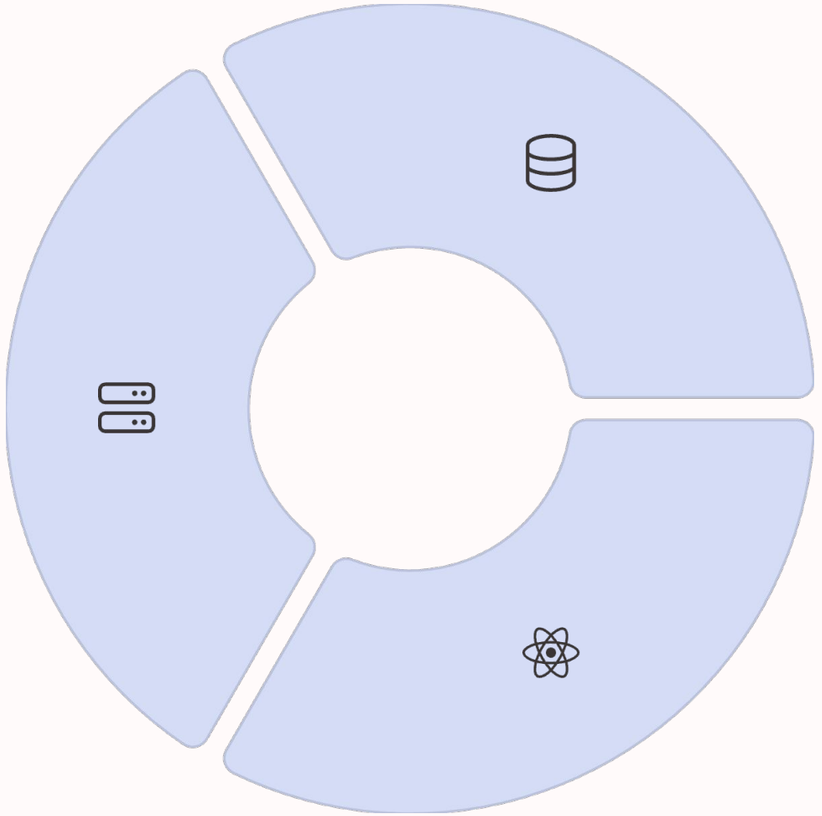
Our infrastructure runs entirely on AWS services.

- EC2 instances for application hosting
- Load balancer for traffic distribution
- PostgreSQL database for data persistence

Technology Stack

Backend

- FastAPI v0.104.1 & Python
- Uvicorn v0.24.0
- Pydantic v2.4.2
- boto3 v1.28.78



Database

- PostgreSQL
- SQLAlchemy v2.0.23
- SQLAlchemy-Utils v0.41.2

Frontend

- React v18.2.0 & TypeScript
- Vite v5.0.8
- TailwindCSS v3.3.6
- Framer Motion v12.12.2

Project Structure

Backend Architecture

```
main.py
app/
    ├── api/routes/
│   ├── account_management.py
│   └── steps.py
├── core/
├── db/
└── services/
```

FastAPI backend follows domain-driven design with clear separation of concerns.

Frontend Organization

```
src/
  ├── components/
  │   ├── agent/
  │   ├── guards/
  │   ├── layout/
  │   ├── migration/
  │   └── ui/
  ├── context/
  ├── data/
  ├── pages/
  ├── services/
  └── types/
```

React frontend uses component-based architecture with TypeScript for type safety.

Database & API Architecture

Database Schema

migration_process	Tracks overall migrations
phase	Migration phases
step	Individual migration steps
step_execution	Execution history
account_management	AWS credentials

Key API Endpoints

- POST /api/account-management
- GET /api/account-management
- GET /api/{phase_type}/{step_slug}
- GET /api/{phase_type}/{step_slug}/history

RESTful API design with clear resource paths and consistent response formats.

Key Features

AWS Account Management

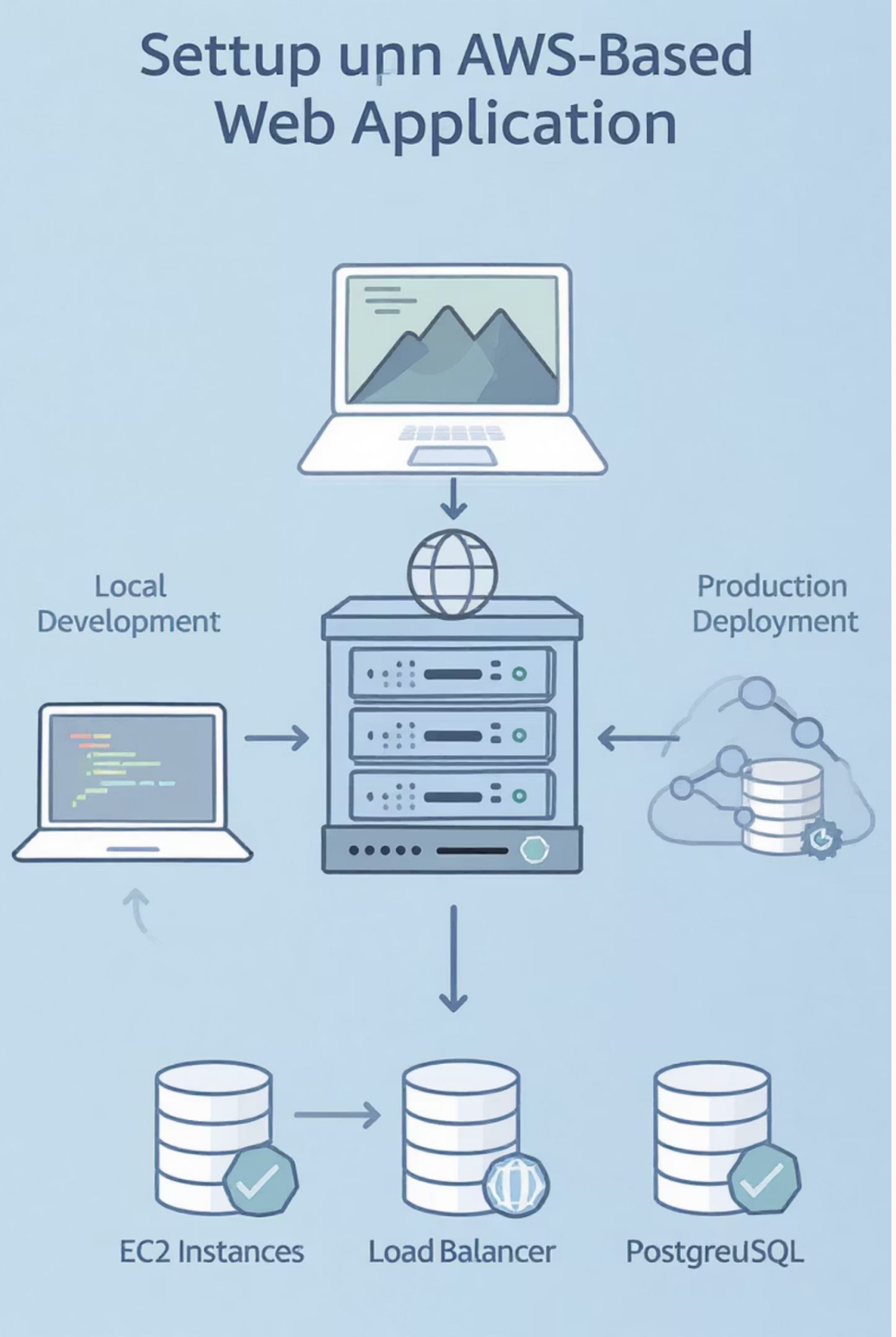
- Add, update, delete AWS credentials
- Test AWS connectivity

Assessment Tools

- Check RAM shared resources
- Identify delegated admin services
- Analyze Cost Explorer data

Preparation & Execution

- Create fallback IAM admin users
- Track migration progress
- View step execution history



Setup & Deployment

Local Setup

Prerequisites: Python 3.8+, Node.js 16+, PostgreSQL, AWS CLI

Environment Config

Set up virtual environment, install dependencies, configure env variables

Production Deploy

Build frontend, configure production server with Gunicorn, set up Nginx

Current production environment uses EC2 instances with Load Balancer and PostgreSQL database.

Migration Phases

1 Assess Existing

Analyze current AWS environment thoroughly.

- Identify resources shared via RAM
- Evaluate delegated admin services
- Audit IAM policies and dependencies

2 Prepare New

Configure destination environment.

- Set up account structure
- Establish proper permissions
- Configure networking foundations

3 Verify New

Validate migration success with comprehensive testing.

- Test functionality
- Verify service configurations
- Confirm resource integrity

4 Migration

Execute resource transfer between organizations.

- Transfer data assets
- Move configured resources
- Implement cutover procedures

5 Attach/Detach

Manage account relationships via AWS Console.

- Access Organizations dashboard
- Execute attachment/detachment operations
- Confirm changes in migration tool

6 Post Migration

Optimize newly migrated environment.

- Clean up temporary resources
- Implement monitoring solutions
- Document final architecture

AWS Services Used in Migration Phases

1 Assess Existing

Analyze current AWS environment with RAM, Organizations, and IAM.

- Identify shared resources and delegated admin services
- Analyze cost data and scan policy documents

2 Prepare New

Configure foundation with IAM, Organizations, and Control Tower.

- Enable MFA and set up billing
- Create OUs and configure SSO with external IdP

3 Migrate & Verify

Transfer accounts and validate configurations.

- Move accounts between organizations
- Test SSO and update billing information

4 Post Migration

Clean up resources and optimize governance.

- Delete legacy CUR reports
- Import accounts into Control Tower

Each phase leverages specific AWS services to ensure smooth migration between organizations. Our tool orchestrates these services through automated workflows.

Migration Workflow Example

1 Login & Account Selection

User authenticates and selects the target AWS account for migration.

2 Journey Overview

Migration phases display with completion status and dependency relationships.

3 Automated Execution

System runs API calls against AWS, displaying real-time logs and results.

4 Manual Steps

Clear instructions guide users through required manual interventions.

5 Progress Tracking

Dashboard updates in real-time showing completed and pending steps.

6 AI Assistance

AI chatbot offers contextual help and can trigger automated actions.

Every step is audited and stored in the step_execution table for comprehensive migration history.

Interactive Demo: Migration Tool Interface

Dashboard View

Migration status at a glance with color-coded progress indicators.

Step Execution

Automated workflows with real-time logs and validation checks.

AI Assistant

Contextual recommendations and automatic error resolution.

The demo showcases our intuitive interface for seamless AWS organization migrations. Watch as we migrate an account through all phases with minimal manual intervention.