SharedSkillet.com: AWS Architecture and 8-Week Implementation Roadmap

AI-Powered Recipes, Video Hosting, and Delivery Services

Nahiyan Bin Noor

Shared Skillet

March 4, 2025

Agenda

Project Context

Key Points:

- **Domain:** Purchased `sharedskillet.com` via Porkbun.
- **Content:**
  - 100 recipe videos (hosted on Google Drive, to be moved to AWS).
  - Text-based recipes (ingredients and instructions).
- **Goal:** Build a platform that integrates:
  - Recipe presentation (text + video).
  - AI-driven image recognition (user uploads dish images).
  - AI-generated recipe instructions & grocery lists.
  - On-demand food delivery within 50 miles.

Core AWS Services

AWS Components to Consider:

- **Route 53** (DNS)
- **S3** (static website hosting, video storage)
- **CloudFront** (CDN)
- **Amplify** (optional for front-end deployment)
- **EC2/ECS/Lambda** (compute)
- **RDS/DynamoDB** (database)
- **Amazon Rekognition** or **SageMaker** (image classification)
- **AWS Bedrock / SageMaker + HF** (LLM-based recipe gen)
- **Amazon Pay** or 3rd party (payment)
- **AWS WAF/IAM** (security)

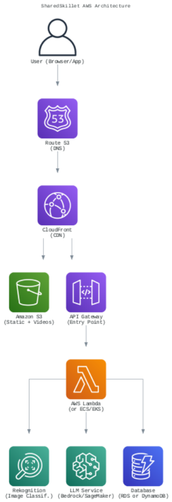
Domain & DNS Setup

Steps to Integrate Porkbun Domain with AWS:

1. In **Route 53**, create a *Public Hosted Zone* for `sharedskillet.com`.
2. Update **Nameservers** on Porkbun to point to those from Route 53.
3. **Configure DNS records** (A, CNAME) for:
    - `www.sharedskillet.com` (front-end)
    - `api.sharedskillet.com` (back-end / API)

Result: Route 53 manages your domain, simplifying subdomain setup and SSL certificates via AWS Certificate Manager.

Infrastructure Overview



(High-level representation of AWS services and data flow.)

Infrastructure Overview: Highlights

Key Points:

- S3 stores videos and static site assets.
- CloudFront delivers content globally with caching & SSL.
- Compute layer (Lambda / ECS) hosts the business logic.
- Rekognition or SageMaker for image recognition.
- RDS / DynamoDB for structured data.

Front-End Hosting & Deployment

Options:

- **S3 + CloudFront**:
  - Host static site in an S3 bucket.
  - Serve via CloudFront for fast global access.
  - Use AWS Certificate Manager for HTTPS.
- **AWS Amplify**:
  - Simplified CI/CD from Git repositories.
  - Great for quick iteration on front-end changes.

Result:

- High availability, low latency, secure delivery of your website.

Back-End & API Layer

Core Responsibilities of the API:

- Serve recipe data (from DB) to front-end.
- Handle user uploads (food images).
- Communicate with AI/ML services for classification and generation.
- Process delivery and payment logic.

Possible Approaches:

1. **Serverless (AWS Lambda + API Gateway)** for on-demand scalability.
2. **ECS/EKS** for container-based workloads (if you need more customization).

Security & Authorization:

- Use **AWS Cognito** or a custom JWT solution to secure your APIs.

Database Considerations

Data Storage Options:

- **Amazon RDS (MySQL/Postgres)** for structured, relational data (recipes, orders, users).
- **Amazon DynamoDB** for flexible, NoSQL use cases.

Typical Schema:

- **Recipes Table:**
  (ID, Name, Ingredients, Instructions, Video URL, Price)
- **Orders Table:**
  (OrderID, UserID, RecipeID, TotalCost, DeliveryAddress, Status, Timestamp)

Key Points:

- Consider **Serverless Aurora** for a pay-per-use relational model.
- Use **AWS Glue** if you need data transformations at scale.

AI/ML: Food Image Recognition

Objective: User uploads a picture of a dish; AI identifies it and returns the matching recipe.

- **Amazon Rekognition:**
  - Pre-trained to recognize many items.
  - Use *Custom Labels* if your recipes are unique or specialized.
- **SageMaker (Custom Model):**
  - Train a CNN or Vision Transformer on your proprietary dataset.
  - More control & potentially higher accuracy if your dishes are very specific.

Flow:

1. User → Upload image → S3 or direct to Lambda.
2. Lambda → Rekognition/SageMaker → returns top matches.
3. Match found → Fetch recipe from DB → show to user.

AI/ML: Recipe Generation & Grocery Lists

Use Cases:

- **Expanding or refining existing recipes.**
- **Auto-generating instructions for recognized dishes not in your DB.**
- **Dynamic grocery list generation**, including substitutes (e.g., dairy-free).

Implementation:

- **AWS Bedrock** for easy access to curated foundation models.
- **SageMaker + Hugging Face** to host open-source LLMs (GPT-J, Llama, etc.).

Integration:

1. API calls an LLM with your base recipe or dish name.
2. LLM returns structured instructions or ingredient lists.
3. Front-end presents these details to the user.

Delivery & Payment Integration

Delivery Radius Logic:

- **Amazon Location Service** (or similar) to compute distance from kitchen to user.
- If distance $\leq$ 50 miles, enable “Delivery” option, else “Pickup” or standard recipe display.

Payment Handling:

- **Amazon Pay** or **Stripe/PayPal**.
- Secure payment checkout with real-time order creation in DB.

Dynamic Pricing:

- **Base Cost** + (Distance-based fee).
- Display final cost on front-end; user can confirm & pay.

Security & Best Practices

Key Focus Areas:

- **IAM Roles**: Use least-privileged roles for each service.
- **HTTPS everywhere**: CloudFront + AWS Certificate Manager for SSL/TLS.
- **WAF (Web Application Firewall)**: Protect from common exploits (SQL injection, XSS).
- **Audit & Logging**: CloudWatch logs, CloudTrail for API calls.

Data Management:

- Store sensitive data (e.g., API keys, DB credentials) in **AWS Secrets Manager**.
- Use **KMS** (Key Management Service) for encryption where needed.

8-Week Roadmap: Overview

Goal: A functional Minimum Viable Product (MVP) with:

- Recipe display (text + video).
- AI-based dish recognition & recipe generation.
- Grocery list creation.
- Delivery + payment workflow.

Approach: Break down into **weekly sprints**:

1. Planning, Setup
2. Data Migration
3. Front-End + Basic API
4. AI Image Recognition (Phase 1)
5. AI Recipe Generation
6. Delivery & Payment Integration
7. Testing & Security
8. Final Launch

Week 1: Planning & Foundation

Tasks:

- Finalize MVP features & scope.
- Set up AWS account, IAM roles, billing alerts.
- Point domain (`sharedskillet.com`) to Route 53.
- Organize recipe dataset (text + videos) from Google Drive.

Outcome:

- Clear scope & AWS environment ready.
- DNS properly configured for `sharedskillet.com`.

Week 2: Data & Video Migration

Tasks:

- Create S3 buckets (one for static front-end, one for videos).
- Upload 100 recipe videos to S3.
- Choose **RDS** or **DynamoDB**, create DB instance.
- Import recipe text data (standardize fields: name, ingredients, instructions).

Outcome:

- S3 hosts all videos.
- DB fully populated with recipe data.

Week 3: Front-End MVP & Basic API

Tasks:

- Initialize a React/Vue/Angular front-end or AWS Amplify project.
- Create pages:
  - Recipe Listing
  - Recipe Detail (embedded video)
- Back-End with **API Gateway + Lambda** or ECS for:
  - `GET /recipes`
  - `GET /recipes/:id`
- Deploy front-end (S3/CloudFront or Amplify).

Outcome:

- Users can browse recipes on `sharedskillet.com`.
- Basic back-end fetch of recipe data.

Week 4: AI Food Image Recognition

Tasks:

- Set up **Amazon Rekognition** (Custom Labels if needed).
- Front-end: Image upload form for dish photos.
- Lambda calls Rekognition, returns predicted dish name & confidence.
- Match dish to recipe ID in DB & display to user.

Outcome:

- Users can upload a photo and receive the matching recipe (if recognized).
- Early testing of AI classification accuracy.

Week 5: AI Recipe Generation & Grocery Lists

Tasks:

- Integrate **LLM** via AWS Bedrock or SageMaker (Hugging Face).
- Allow AI to generate or refine recipe steps (if missing in DB).
- Automatically compile grocery lists from recipe ingredients.
- Front-end UI for users to view/print grocery lists.

Outcome:

- Dynamic recipe instructions for recognized dishes or textual requests.
- Clear grocery lists (with potential substitutions).

Week 6: Delivery & Payment Integration

Tasks:

- Implement distance check (≤ 50 miles) for delivery using location services.
- Dynamic pricing: base cost + distance fee.
- Integrate **Amazon Pay** or Stripe for payment.
- Create an `Orders` table and store order details.

Outcome:

- Users can order a prepared dish if they're within 50 miles.
- Secure checkout flow & order confirmation.

Week 7: Testing, Security, & Feedback

Tasks:

- **End-to-end testing:** image upload, recipe generation, order placement.
- Add or refine **WAF** rules, confirm HTTPS with AWS Cert Manager.
- Limited **Beta** for real-user feedback (friends, family, small group).
- Collect metrics (CloudWatch, logs) for performance usage.

Outcome:

- Validated MVP with essential security measures.
- Bug and user-experience reports for final fixes.

Week 8: Final Adjustments & Launch

Tasks:

- Fix priority issues discovered in beta.
- Polish UI/UX (styling, layout, instructions).
- Confirm cost optimizations (auto-scaling, usage, budgets).
- Publicly launch `sharedskillet.com`, announce to broader audience.

Outcome:

- Fully functional MVP is live and ready for real users.
- Foundation in place for future enhancements (mobile app, community features).

Next Steps & Future Enhancements

Potential Growth Areas:

- **Multilingual Support**: Automatic translation of recipes & instructions.
- **User Community**: Ratings, reviews, user-submitted recipes.
- **Analytics & Reporting**: Amazon QuickSight for advanced insights.
- **Mobile App**: Native iOS/Android integration with the same AWS back-end.
- **Subscription Model**: Premium AI features, free delivery tiers, exclusive recipes.

Summary

Key Takeaways:

- AWS offers a comprehensive solution for hosting, AI/ML, and scalability.
- An 8-week plan provides a clear path from zero to MVP.
- Focus on security, cost monitoring, and user feedback loops.

Thank you!