Cloudelligent

Migration Plan: Crema.co (Heroku to AWS)

Client: Crema.co
Document Ref: MIGRATION-PLAN-V1

1. Executive Summary

This document outlines the technical execution plan for migrating the Crema.co Ruby on Rails e-commerce platform from Heroku to Amazon Web Services (AWS). The objective is to transition the monolithic application to a scalable, containerized architecture using AWS ECS Fargate, Amazon RDS, and ElastiCache, while ensuring minimal downtime and robust security via private networking (NAT Gateways) and Application Load Balancers (ALB).

2. Prerequisites & Access Verification

Before initiating the migration, the following access points have been confirmed:

- **Heroku:** Full access to the existing production application and configuration.
- **GitHub:** Access to the master branch containing the source code and Dockerfile.
- **AWS:** Administrator access to the target AWS account for resource provisioning.

3. Target Architecture Overview

Component	Strategy	Details
Compute	**ECS Fargate**	Serverless container execution. Eliminates need for EC2 instance management.
Database	**Amazon RDS**	PostgreSQL instance (provisioned to match or exceed Heroku spec).
Caching	**ElastiCache**	Redis Cluster mode enabled (if required) or Primary/Replica setup.
Networking	**VPC with NAT**	**Public Subnets:** ALB, NAT

		Gateways. **Private Subnets:** ECS Tasks, RDS, ElastiCache.
Load Balancing	**ALB**	Application Load Balancer handling SSL termination and traffic distribution.

4. Technical Migration Phases

Phase 1: Application Containerization & Configuration

Objective: Finalize the Docker image and secure application configuration.

1. **Dockerfile Finalization:**
   - Utilize the Dockerfile located in the master branch.
   - Apply the necessary tweaks identified during initial testing.
   - **Action:** Build and tag the image locally to verify the build process completes successfully.
2. **Environment Variable Retrieval (Critical Path):**
   - *Current Issue:* Local tests are failing due to missing environment variables.
   - **Storage:** Upload sensitive variables to **AWS Secrets Manager** or **AWS Systems Manager Parameter Store**.

Phase 2: Infrastructure Provisioning (IaC)

Objective: Provision the "empty" environment ready to receive data and code.

1. **Network Setup:**
   - Create VPC.
   - Deploy **Internet Gateways** for public subnets.
   - Deploy **NAT Gateways** in public subnets to allow private ECS tasks to pull images and reach external APIs.
2. **Database & Cache:**
   - Provision **RDS for PostgreSQL** (20GB to start to accommodate the 10GB data + growth).
   - Provision **ElastiCache for Redis**.
3. **Compute Resources:**
   - Create ECS Cluster.
   - Create ECR Repository.
   - Configure Security Groups:
     - *ALB SG:* Allow 443 from 0.0.0.0/0.

- *App SG:* Allow traffic only from ALB SG.
- *DB/Redis SG:* Allow traffic only from App SG.

Phase 3: CI/CD Pipeline Implementation

Objective: Automate the build and deployment process to eliminate manual errors and ensure consistent releases.

1. **Source Stage:**
   - Connect **AWS CodePipeline** to the GitHub repository.
   - Configure a webhook to trigger the pipeline on commits to the master branch.
2. **Build Stage (AWS CodeBuild):**
   - Create a buildspec.yml file (if not present) to handle the build logic.
   - **Steps:** Log in to ECR, build the Docker image, tag the image with the commit SHA, and push the image to ECR.
   - **Output:** Generate an imagedefinitions.json file for the deploy stage.
3. **Deploy Stage (ECS):**
   - Configure CodePipeline to deploy to **Amazon ECS**.
   - Use the imagedefinitions.json artifact to update the ECS Service with the new image.
   - Configure rolling updates to ensure zero downtime during deployments.

Phase 4: Scheduled Tasks Implementation (Heroku Scheduler Replacement)

Objective: Migrate recurring jobs (Rake tasks) from Heroku Scheduler to AWS Native services.

Strategy: Use **Amazon EventBridge Scheduler** to trigger standalone Fargate tasks using the "Container Override" pattern.

Implementation Details:

1. Dual Task Definition Strategy:
   To optimize resources and security, we will decouple the scheduled tasks from the main service.
   - **Task Definition 1 (crema-app):** Runs long-lived Web and Sidekiq services.
   - **Task Definition 2 (crema-rake-tasks):** Runs ephemeral scheduled tasks. This allows for tighter IAM permissions (only needs permissions relevant to the specific rake tasks) and different resource sizing (often lower CPU/Memory requirements than the web app).
2. **Task Definition Configuration (crema-rake-tasks):**
   - **Network Mode:** awsvpc (Required for Fargate).
   - **Image:** Same ECR URI as the main app (latest).
   - **Command:** Set a default placeholder command (e.g., ["echo", "EventBridge

Override Placeholder"]). This safeguards against accidental execution without overrides.

- **Roles:**
  - executionRoleArn: For pulling images and fetching Secrets (SSM/Secrets Manager).
  - taskRoleArn: Permissions required by the Rails app itself (e.g., S3 access).
- **Configuration Reference (JSON):**

```
{
 "family": "crema-rake-tasks",
 "networkMode": "awsvpc",
 "requiresCompatibilities": ["FARGATE"],
 "cpu": "256", "memory": "512",
 "containerDefinitions": [
  {
   "name": "rake-runner",
   "image": "<YOUR_ECR_URI>:latest",
   "command": ["echo", "Command overridden by EventBridge"],
   "secrets": [
    {"name": "DATABASE_URL", "valueFrom": "arn:aws:ssm..."},
    {"name": "REDIS_URL", "valueFrom": "arn:aws:ssm..."}
   ],
   "logConfiguration": {
     "logDriver": "awslogs",
     "options": {
       "awslogs-group": "/ecs/crema-rake-tasks",
       "awslogs-stream-prefix": "rake"
     }
   }
  }
 ]
}
```

3. **EventBridge Rules Setup:**
   - **Source:** Audit Heroku Scheduler for active jobs (e.g., rake daily:report, rake subscription:renew).
   - **Schedule:** Create EventBridge rules using Cron expressions matching the Heroku frequency.
   - **Target:** ECS Cluster -> Task Definition: crema-rake-tasks.
   - **Container Overrides:** For each rule, configure the "Container Overrides" to inject the specific Rake command.

        - Command: ["bundle", "exec", "rake", "subscription:renew"]
4. **End-to-End Flow Example (Daily Renewals):**
    - **Trigger:** At **09:00 UTC**, the EventBridge Rule crema-daily-renewals activates.
    - **Dispatch:** It invokes the **RunTask** API on the crema-ecs-cluster.
    - **Override:** It loads the crema-rake-tasks definition but **replaces** the default echo command with ["bundle", "exec", "rake", "subscription:renew"].
    - **Execution:** Fargate provisions a container, runs the specific rake task, and streams output to CloudWatch Logs (/ecs/crema-rake-tasks/rake/...).
    - **Completion:** The task finishes successfully and the container terminates automatically (saving costs compared to a constantly running worker).

Phase 5: Data Migration (Big Bang Strategy)

Objective: Migrate the 10GB PostgreSQL database with data consistency.

- **Assumption:** A "Big Bang" approach is approved. This implies a scheduled maintenance window where the site is read-only or offline.

1. **Preparation:**
    - Notify stakeholders of the maintenance window.
2. **Execution (Maintenance Window):**
    - **Stop Traffic:** Enable maintenance mode on Heroku to stop writes.
    - **Export:** Create a fresh dump of the Heroku Postgres database.
    - **Import:** Restore the dump to the new Amazon RDS instance using pg_restore.
    - **Validation:** Verify row counts on critical tables (Users, Orders, Subscriptions).

Phase 6: Redis Strategy

Objective: Stand up the cache layer.

- **Strategy:** Fresh Start.
- **Assumption:** As confirmed, Redis is used strictly as a cache/session store. No data migration is required.
- **Action:** Ensure the Rails config/cable.yml and config/redis.yml (or ENV vars) are updated to point to the new ElastiCache endpoint. Old sessions will be invalidated, requiring users to log in again post-migration.

Phase 7: Deployment & Cutover

Objective: Go live on AWS.

1. **Deploy Application:**
    - Trigger the manual release or push to master to initiate the new CI/CD pipeline.
    - Wait for Health Checks to pass on the Target Group.
2. **Testing:**

- Access the application via the ALB DNS name.
- Verify connectivity to RDS and Redis.
- Test critical flows: Login, View Subscription, Checkout (using test mode if possible, or verify Stripe API connectivity).

3. **DNS Cutover:**
   - Update Route53 (or current DNS provider) to point the domain to the AWS ALB.
   - Provision SSL Certificates via ACM (AWS Certificate Manager) attached to the ALB.