Atlas Engine Solution Design Document

AI-Powered Sales Acceleration Platform Solution Design Document

Prepared By: Michael Weed

Version: v1

Date: November 6, 2025

Classification: Public

Executive Summary

- **Problem Statement -** In today's market, the speed of lead response is a primary driver of conversion. Most businesses struggle with "leaky" sales funnels due to slow manual follow-up, gaps in 24/7 coverage, and inconsistent CRM data entry.
- **Solution Overview -** The Atlas Engine is an open-source, AI-powered sales acceleration platform. It provides a "production-grade" foundation to solve this problem by orchestrating a 24/7/365 conversational AI (via chat or voice) that intelligently captures lead information, generates dynamic responses, and automatically creates or updates leads in Salesforce—all within minutes of the initial contact.
- **Important Distinction -** This solution is **"production-grade," not "production-ready."** It is a robust, secure, and scalable framework built on serverless best practices. To become "production-ready," any adopting organization must perform mandatory customizations to tailor the conversational logic (e.g., Lex intents, Lambda verbiage) to their specific business purpose.
- **Expected Benefits:**
    - Dramatically reduce lead response time from days or hours to minutes.
    - Eliminate manual data entry errors by syncing 100% of interactions to the CRM.
    - Provide 24/7/365 "always-on" lead capture and qualification.
    - Democratize access to an enterprise-grade AI sales automation solution.

---

1. Introduction

1.1 Purpose

This document provides a comprehensive architectural and technical overview of the Atlas Engine. It is intended for AWS and Salesforce architects, developers, and business stakeholders who wish to adopt, customize, and deploy the solution.

1.2 Scope

This document details the architecture and design of the Atlas Engine as published on GitHub and the AWS Serverless Application Repository (SAR).

- **In Scope:**
  - Conversational AI front-end using Amazon Lex (chat and voice).
  - Backend orchestration using AWS Step Functions.
  - Core business logic via 8 AWS Lambda functions.
  - Data persistence using 2 Amazon DynamoDB tables.
  - Generative AI for dynamic responses via Amazon Bedrock.
  - CRM integration (create/update Leads) with Salesforce.
  - Telephony for *outbound* calls initiated by the workflow via Amazon Connect.
- **Out of Scope:**
  - **Inbound Call Flows -** The solution does not provide pre-configured Amazon Connect *inbound* contact flows, queues, or agent dashboards.
  - **Advanced CRM Automation -** The solution's responsibility ends at creating/updating the Lead in Salesforce. It does not include Salesforce-native automation (e.g., Apex, Flows) for lead assignment or alerting.
  - **Complex Entity Disambiguation -** The solution is configured to check for existing *Leads*. It does not handle the complex logic of disambiguating between Leads, Contacts, or multiple Accounts.
  - **Additional Channels -** The base solution does not include native integration for SMS, WhatsApp, or Email.

2. Business Context and Objectives

2.1 Strategic Alignment

The core strategy of the Atlas Engine is to democratize access to a real, working AI chat/voice solution. It provides a robust, scalable, and secure foundation, allowing organizations to bypass months of development and focus immediately on customizing the solution to their unique business needs.

2.2 Key Business Objectives

Objective	Metric	Target
Reduce Lead Response Time	Time from web inquiry to CRM entry.	< 1 minute
Improve Data Hygiene	% of automated interactions logged in CRM.	100%
Increase Lead Capture	% of off-hours inquiries captured.	100%
Reduce Manual Workload	Manual data entry hours per sales rep.	Eliminate

---

3. Solution Architecture

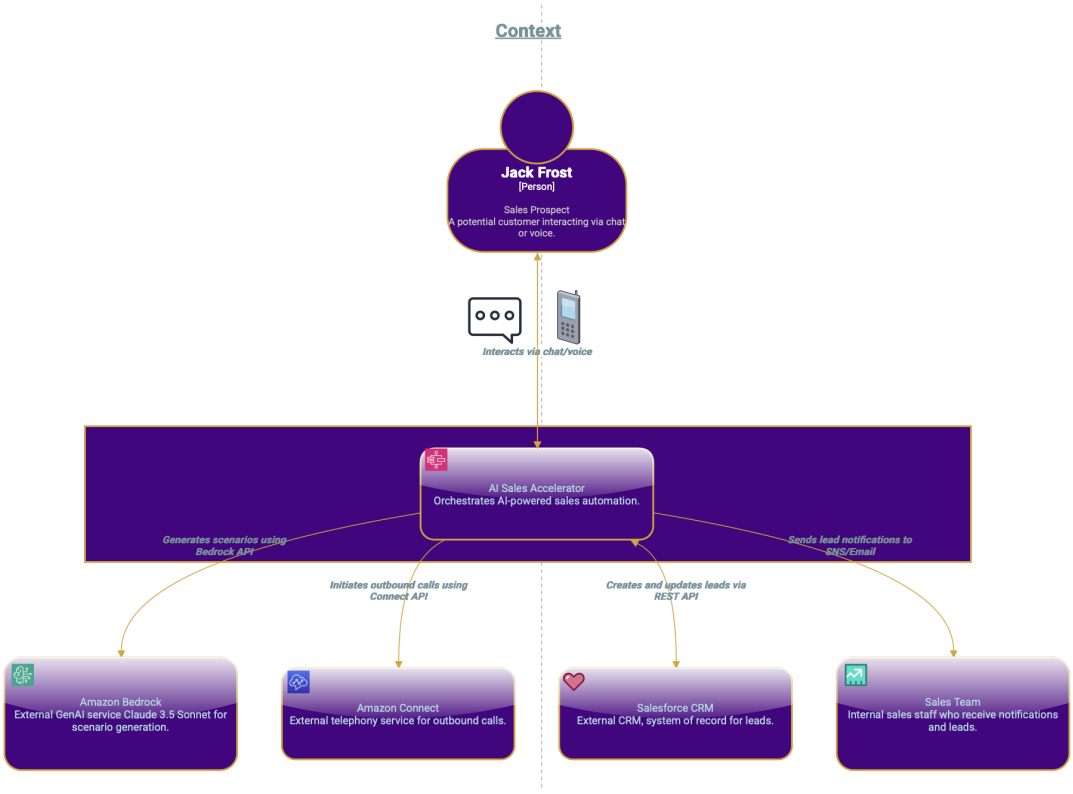
3.1 Architectural Overview

The Atlas Engine is a serverless, event-driven architecture. The entire business process is orchestrated by an AWS Step Functions (sfn) workflow, which ensures the process is auditable, durable, and resilient.

This orchestration-based design is a critical design choice. It decouples the "front-end" conversational AI (Amazon Lex) from the "back-end" business logic (Salesforce, Bedrock, Connect). This allows the Lex bot to respond to the user almost instantly ("Thanks, we'll process that") while the long-running, multi-step workflow executes in the background.

A Note on the Implementation: *A significant portion of the initial boilerplate code for the Lambda functions was generated using AI. This approach proved to be a powerful accelerator. This AI-generated baseline was then rigorously hand-tuned, refined, and tested—with over 300 full end-to-end state machine executions—to prove that AI-generated code, in the right hands, can be a reliable and powerful tool for building production-grade solutions.*

3.2 High-Level Architecture Diagram



3.3 Core Workflow

1. A **User** interacts with **Amazon Lex** (chat/voice).
2. Lex captures all required information (slots) and invokes the **`LexFulfillmentHandler` (Lambda)**.
3. The `LexFulfillmentHandler` passes the user's data to the **AWS Step Functions** workflow and immediately returns a confirmation to the user.
4. The **Step Function** begins its 4-phase orchestration: a. **Phase 1: `CreateLeadHandler`** - Connects to Salesforce, performs an idempotent check to see if the Lead exists, and then creates or updates the Lead record. b. **Phase 2: `GenerateDynamicScenarioHandler`** - Takes the conversation context and uses **Amazon Bedrock** to generate a personalized script for an outbound call. c. **Phase 3: `InvokeOutboundCallHandler`** - Places the outbound call via **Amazon Connect** using the AI-generated script. It uses a `taskToken` to pause the workflow until the call is completed. d. **Phase 4: `UpdateLeadHandler`** - Updates the Salesforce Lead with the call's summary and outcome.

3.4 Key Design Decisions

Decision	Rationale	Alternatives Considered
Serverless-First (Lambda, sfn, DynamoDB)	For near-infinite scalability, zero infrastructure management, and a pay-per-use cost model.	EC2-based services would require auto-scaling, patching, and higher idle costs.
AWS SAM for Deployment	Provides a "one-click" (or `deploy.sh`) deployment, simplifying setup and enabling easy environment-based customization.	Manual deployment via the console is error-prone, not repeatable, and not scalable.

Step Functions (Standard) for Orchestration	Provides a durable, auditable, and long-running (up to 1 year) workflow. Critically, it decouples the fast-responding Lex bot from the slower, multi-step backend process.	A single "fat" Lambda would be brittle, have a 15-min timeout, and would not be auditable.
Idempotent Lead Creation	The `CreateLeadHandler` queries Salesforce before it writes.	A naive "create" operation would generate duplicate leads every time the Step Function retried a failed step.
Centralized Credential Management	All Salesforce credentials (JWT key, client ID, username) are stored in AWS Secrets Manager.	Hardcoding secrets in Lambda environment variables or in the code itself is a critical security vulnerability.

---

4. Technology Stack

4.1 Proposed Stack

Layer	Technology	Count	Purpose
Deployment	AWS SAM CLI, CloudFormation	1	Infrastructure as Code (IaC) for automated, repeatable deployments.
Compute	AWS Lambda (Python 3.13)	8	Core business logic (e.g., `CreateLeadHandler`, `GenerateScenarioHandler`).
Orchestration	AWS Step Functions (Standard)	1	The "brain" that coordinates the multi-step sales workflow.
Database	Amazon DynamoDB Tables	2	`AtlasEngineInteractions` (state) and `AtlasEngineTaskTokens`(callback state).
Conversational AI	Amazon Lex V2	1	NLU/NLG for chat and voice interaction.
Generative AI	Amazon Bedrock (Claude 3.5 Sonnet)	1	Generates dynamic, context-aware call scripts.
CRM	Salesforce (via REST API)	1	The external system of record for Leads.

Telephony	Amazon Connect	1	Handles the programmatic `start_outbound_voice_contact` API call.
Security	AWS Secrets Manager, AWS IAM	1, 8+	Secure credential storage and least-privilege permissions.
Libraries	AWS Lambda Layers	2	`simple-salesforce` (CRM), `PyJWT` (Auth), `requests`, `phonenumbers`.

4.2 A Note on the Lambda Code

As mentioned in the architecture section, the Lambda code is a powerful example of an AI-assisted workflow. The initial function scaffolding and logic were generated using AI, which was then heavily refined and rigorously tested (over 300 full-flow executions) to build the final, production-grade functions. This project is proof that AI-generated code *can* be a powerful tool in the right hands.

---

5. Data Model and Flows

1. **Lex Input -** Lex passes a JSON `event` to the `LexFulfillmentHandler` containing the user's "slots" (e.g., name, email, phone).
2. **Workflow Input -** The Lambda passes a filtered JSON object as `input` to the Step Function workflow.
3. **Salesforce (Write) -** The `CreateLeadHandler` maps this input to a Salesforce `Lead` SObject. It performs a `SOQL` query (`SELECT Id FROM Lead WHERE Email =...`) before executing a `DML` (create/update) operation. This "query-before-write" is the critical idempotency check.
4. **DynamoDB (State):**
    - `AtlasEngineInteractions` stores conversation history and state.
    - `AtlasEngineTaskTokens` stores the Step Function `taskToken`, allowing the workflow to pause indefinitely and be resumed by an external service (Amazon Connect).

---

6. Integration Strategy

- **Salesforce (JWT Flow)** integration is via the Salesforce REST API using the OAuth 2.0 JWT Bearer Flow for server-to-server integration. The `CreateLeadHandler` builds a JWT, signs it with the private key (from Secrets Manager), and exchanges it with Salesforce for an access token.
- **Amazon Bedrock (API)** integration is a direct, synchronous `invoke_model` API call from the `GenerateDynamicScenarioHandler` via the AWS boto3 SDK.
- **Amazon Connect (Callback Pattern) i**ntegration uses the `start_outbound_voice_contact` API. The Step Function's `taskToken` is passed to Connect. When the call completes, a separate Lambda (triggered by Connect) uses that token to send a "success" or "failure" signal back to the Step Function, allowing the workflow to resume.

---

7. Security and Compliance

The solution is built with a "security-first" mindset, suitable for enterprise deployment.

- **Threat: Credential Exposure**
    - **Mitigation: AWS Secrets Manager.** All Salesforce credentials (private key, client ID) are stored in Secrets Manager. The `CreateLeadHandler`'s IAM role is the *only* entity with permission to read it at runtime. No credentials are *ever* hardcoded.
- **Threat: Elevation of Privilege**
    - **Mitigation: Least-Privilege IAM Roles.** The solution does *not* use a single, over-permissive "Lambda" role. It creates 8+ granular, single-purpose IAM roles. Each Lambda function has its own role granting *only*the permissions needed for its specific job.
- **Threat: Data-in-Transit**
    - **Mitigation:** All API endpoints (Lex, AWS, Salesforce) use HTTPS.
- **Threat: Data-at-Rest**
    - **Mitigation:** All data in DynamoDB tables is encrypted at rest using AWS-managed keys.

---

8. The Path to "Production-Ready"

Deploying the solution with the `deploy.sh` script or AWS SAR takes ~30 minutes. This creates a "production-grade" system. To make it "production-ready" for a specific organization, the following mandatory steps must be performed.

1. **Phase 1: Customize Conversation (Lex)**
    - **Task:** Navigate to the Amazon Lex console and select the deployed bot.
    - **Action:** Update the `Intents` (e.g., `RequestDemo`) and `Utterances` to match the organization's specific language, products, and services.
2. **Phase 2: Customize Logic (Lambda)**
    - **Task:** Open the Lambda function code, specifically `LexFulfillmentHandler_code/`.
    - **Action:** Update the function's internal "router" to handle the new intents created in Phase 1. Change any hardcoded verbiage (e.g., bot responses, prompts) to match the new conversational design.
3. **Phase 3: Customize Documentation (GitHub/SAM)**
    - **Task:** Update the public-facing documentation.
    - **Action:** Fork the repository and update the `README.md` and AWS SAM page to reflect the new, customized purpose of the bot, ensuring users understand how to interact with it.

---

9. Operating Model

- **Support:** As an open-source project, support is provided via GitHub Issues on the main repository.
- **Monitoring -** The two primary tools for monitoring are:
    1. **AWS CloudWatch Logs -** Use `sam logs --stack-name AtlasEngine-dev --tail` to view real-time logs from all functions.
    2. **AWS Step Functions Console -** Visually inspect failed executions to diagnose which step in the workflow failed and why.
- **Cleanup -** The `cleanup.sh` script is provided to tear down all deployed resources for a specific environment.

---

10. Risks, Warnings, and Mitigation

Risk / Warning	Probability	Impact	Mitigation / Action
High Voice Latency	High	High	**This is the biggest known issue.** There is a delay of up to 10 seconds on voice calls as the Step Function transitions between states. This is a poor customer experience. See Roadmap item 11.4.
Lead/Contact Disambiguation	High	Medium	The solution only checks for existing Leads. If an existing Contact or a user with multiple Accounts calls, the system may create a duplicate Lead. See Roadmap item 11.3.
Deployment Failure (Bedrock)	Medium	High	The `deploy.sh` script will fail if the AWS account has not manually enabled access to the required Bedrock models in the console.

---

11. Future Roadmap and Opportunities

The Atlas Engine is a strong foundation. Adopters are encouraged to explore the following high-value enhancements.

1. **Full Amazon Connect Integration -** The current solution only handles *outbound* calls. A high-value next step is to build the *inbound* Contact Flow, allowing users to call a number and interact with the same Lex bot. This would also involve setting up queues, reports, and agent dashboards.
2. **Salesforce Automation -** Create Salesforce Flows or Apex Triggers that activate when a `Lead` is created by the Atlas Engine. This automation could assign the lead to the correct sales team, send an internal alert, or create a follow-up task. This is a high-value, low-effort objective.
3. **Solve Disambiguation (R-02) -** For enterprise adopters, enhance the `CreateLeadHandler` to gracefully handle disambiguation. This would involve modifying the SOQL to query Contacts and Accounts, not just Leads. This is a high-effort task, as handling conflicts ("John Smith") or multiple accounts for one person breaks the 80/20 rule of this demo.
4. **Resolve High Latency (R-01) -** This is the biggest opportunity for CX improvement.
    - **Architecture -** The state machine could be updated to run some tasks in parallel (e.g., auth and pre-fetching).
    - **Experience -** Implement latency masking strategies in the Connect flow. This involves playing simulated audio (like keyboard typing) or interjecting "Hmm, let me look that up..." to make the wait feel natural.
5. **Omnichannel Expansion (SMS/Email):**
    - **SMS -** Adding SMS to the workflow is a very strong benefit. A single number (via Amazon Pinpoint or Twilio) can often be used for WhatsApp, Telegram, etc., and Lex supports these channels.
    - **Email -** Once SMS is added, adding email (via Amazon SES) is almost a requirement. At this point, you are offering true 24/7/365 services, handling low-value contacts cost-effectively and keeping valuable employees focused on valuable work.

12. Cost Analysis

Costs are entirely variable based on usage. The serverless model ensures there are no idle costs, outside of minimal storage for DynamoDB.

Environment	Monthly Cost	Use Case / Conversations
Development	**~$24/month**	Testing, ~1,000 conversations
Staging	**~$214/month**	QA, ~10,000 conversations
Production	**~$1,138/month**	Live, ~50,000 conversations

Export to Sheets

Note: An estimated detailed cost breakdown is available in the `COST_ESTIMATION.md` file in the GitHub repository.

---

13. Appendices

13.1 Glossary

Throughout this document, we use a few common abbreviations. "SAM" refers to the AWS Serverless Application Model, which is how we deploy the application. "sfn" is our shorthand for the AWS Step Functions orchestrator. When we talk about "SOQL," we're referring to the Salesforce Object Query Language used for database queries.

13.2 References

- **GitHub Repository:** `https://github.com/MichaelWeed/atlas-engine`
- **AWS SAR Page:** `https://console.aws.amazon.com/serverlessrepo/home?region=us-west-2#/applications/arn:aws:serverlessrepo:us-west-2:975049941322:applications~atlas-engine`

Approval

Role	Name	Signature	Date
Architect	Michael Weed		