perplexity

System Architecture: Tech Stack and Directory Structure

Below is an expert breakdown of what a lead generation platform using Apollo.io and Make.com would look like from a system architect's perspective, covering both the tech stack and a recommended directory structure.

Tech Stack Overview

Layer	Tools/Technologies	Purpose
Frontend	React, JavaScript, HTML5, CSS3	Custom dashboards, campaign management UI
Backend	Node.js, Python, Ruby on Rails	API orchestration, business logic, workflow triggers
Automation/Workflow	Make.com (formerly Integromat)	Orchestrates data flows, API calls, and process automation
Lead Data Source	Apollo.io (API, Chrome Extension)	Lead discovery, enrichment, and export
Database/Storage	PostgreSQL, MongoDB, Airtable, Google Sheets	Stores leads, campaign data, engagement logs
CRM Integration	Salesforce, HubSpot, Pipedrive (via API/Make.com)	Syncs enriched leads and engagement data
Email/SMS	SendGrid, Mailgun, Twilio (via Make.com)	Automated outreach and follow-ups
Other Integrations	Zapier, custom webhooks, LinkedIn, Slack	Additional automation and notifications
Containerization/DevOps	Docker, Kubernetes, Terraform	Deployment, scaling, and infrastructure management
Testing/CI	Jest, Cypress, Selenium, GitHub Actions	Quality assurance and automated testing

Apollo.io's tech stack includes Node.js, Ruby on Rails, Python, and more for robust API and integration support [1] [2].

Directory Structure Example

Below is a typical directory structure for such a platform, assuming a monorepo approach for both backend and frontend components, and including automation scripts:

```
/leadgen-platform
|
```

```
├── /frontend/
│   ├── /public/
│   ├── /src/
│   │   ├── /components/
│   │   ├── /pages/
│   │   ├── /services/
│   │   └── App.js
│   └── package.json
│
├── /backend/
│   ├── /api/
│   │   ├── /apollo/
│   │   │   └── apolloClient.js
│   │   ├── /crm/
│   │   │   └── crmSync.js
│   │   └── /email/
│   │       └── emailSender.js
│   ├── /models/
│   ├── /controllers/
│   ├── /services/
│   └── server.js
│
├── /automation/
│   ├── /make/
│   │   ├── apollo_to_crm_scenario.json
│   │   └── lead_enrichment_flow.json
│   ├── /scripts/
│   │   └── data_cleanup.js
│   └── README.md
│
├── /integrations/
│   ├── /webhooks/
│   ├── /zapier/
│   └── /chrome_extension/
│
├── /database/
│   ├── /migrations/
│   ├── /seeds/
│   └── schema.sql
│
├── /docs/
│   ├── architecture.md
│   ├── api.md
│   └── workflows.md
│
├── .env
├── docker-compose.yml
├── README.md
└── package.json
```

Key Architectural Notes

- Apollo.io **API** is used for lead sourcing, enrichment, and segmentation. API calls are managed in `/backend/api/apollo/` and orchestrated via Make.com scenarios[3] [4] [5] [2].
- Make.com scenarios are stored as JSON or documented in `/automation/make/`, handling workflow automation such as syncing leads to CRM, triggering outreach, and updating lead statuses[6] [5].
- **CRM and Email Integrations** are modular, allowing easy swapping or extension to new tools.
- **Database Layer** stores persistent lead data, campaign metadata, and engagement logs for analytics and reporting.
- **DevOps and Containerization** ensure the platform is scalable and maintainable, supporting CI/CD pipelines and infrastructure as code[1].

This architecture enables seamless, automated lead generation and nurturing, with Apollo.io as the data engine and Make.com as the workflow orchestrator, all integrated into a scalable and maintainable system[6] [1] [4] [2].

⁂

1. https://himalayas.app/companies/apollo/tech-stack
2. https://www.uplead.com/how-to-use-apollo-io/
3. https://knowledge.apollo.io/hc/en-us
4. https://www.uplead.com/apollo-io-integrations/
5. https://community.make.com/t/apollo-io-api-workflow/75778
6. https://www.youtube.com/watch?v=y35hLWdtB4Y