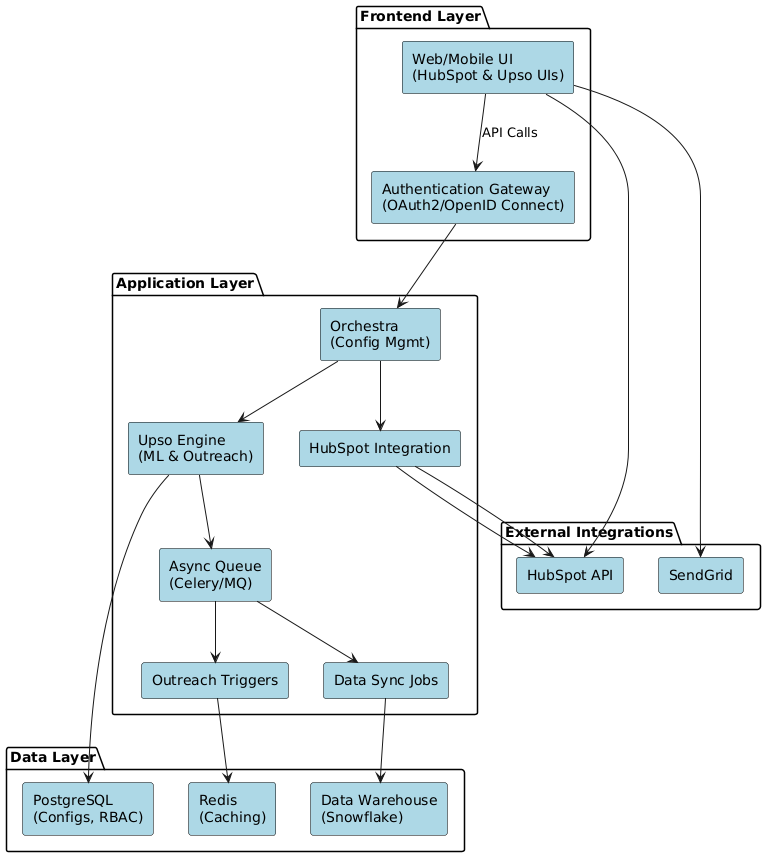
## **Table of Contents**

1. **High-Level Architecture**
2. **Infrastructure Diagram**
3. **Monitoring & Observability**
4. **Programming Languages**
5. **Simplified Design for Low Scale**
6. **Preventing Inconsistent Configurations**
7. **Key Improvements & Security Considerations**

## **1. High-Level Architecture**

### **Core Components**

1. **Frontend Layer**
   * **Web/Mobile UI**: Interfaces for pipeline tracking (HubSpot) and upsell outreach (Upso).
   * **Authentication Gateway**: OAuth2/OpenID Connect (Auth0/Keycloak) for centralized access control.
2. **Application Layer**
   * **Orchestra (Orchestration Layer)**:
     + Central configuration management (prevents direct tool changes).
     + Validates/enforces configurations via APIs.
   * **HubSpot Integration Service**: Syncs CRM data and proxies configuration changes.
   * **Upso Engine**: Analyzes product signals (ML models) and triggers automated customer outreach.
   * **Async Task Queue**: Manages background jobs (email campaigns, data syncs).
3. **Data Layer**
   * **Relational Database**: PostgreSQL (configurations, user roles, audit logs).
   * **Data Warehouse**: Snowflake/BigQuery (analytics on pipelines/customers).
   * **Caching Layer**: Redis (reduces HubSpot API load, speeds Upso processing).
4. **External Integrations**
   * **HubSpot API**: Real-time CRM data sync.
   * **Email/SMS APIs**: SendGrid/Twilio for outreach.



## **2. Infrastructure Diagram (Cloud-Native)**

### **Cloud Provider: AWS/GCP/Azure (multi-region deployment).**

* **Global Load Balancer**: Routes traffic across regions (e.g., US-East, EU-Central).
* **Compute**:
  + **Kubernetes Clusters**: Host stateless services (Orchestra, Upso) in regional pods.
  + **Serverless Functions**: Handle HubSpot webhooks and lightweight tasks.
* **Database**:
  + **Managed PostgreSQL**: Multi-AZ with read replicas and cross-region backups.
* **Networking**:
  + VPCs with private subnets for internal services.
  + API Gateway for secure HubSpot/SendGrid integrations.
* **Backup & Recovery**:
  + Daily automated snapshots + disaster recovery replication.

## **3. Monitoring & Observability**

### **Tools**

* **Infrastructure**: AWS CloudWatch, GCP Stackdriver.
* **Application Performance**: New Relic/Datadog (end-to-end API tracing).
* **Logging**: ELK Stack (Elasticsearch, Logstash, Kibana) for aggregated logs.
* **Metrics**: Prometheus + Grafana (track API latency, error rates, queue health).

### **Practices**

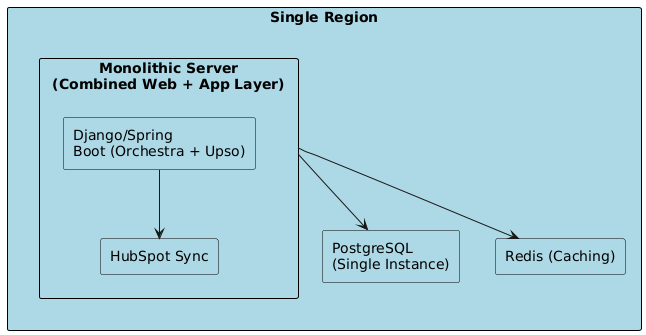
* **SLOs/SLAs**: Define uptime targets (e.g., 99.9% for configuration APIs).
* **Automated Alerts**: Escalate via PagerDuty/Opsgenie for critical issues (e.g., failed HubSpot syncs).
* **Synthetic Checks**: Monitor uptime of key endpoints (e.g., Upso email triggers).

## **4. Programming Languages**

| **Component** | **Language** | **Rationale** |
| --- | --- | --- |
| Orchestra (Orchestration) | Go | High concurrency, low latency for API-heavy workflows. |
| Upso Engine | Python | Rich ML libraries (scikit-learn) for customer focused models. |
| Frontend | React + TypeScript | Type safety for complex UIs (pipeline dashboards). |
| Infrastructure-as-Code | Terraform | Standard for cloud-agnostic provisioning (AWS/GCP/Azure). |

## **5. Simplified Design for Low Scale (<10 Companies)**

1. **Serverless Architecture**:
   * Use AWS Lambda/Azure Functions for Orchestra/Upso (no server management).
   * Single PostgreSQL instance (AWS RDS) with nightly backups.
2. **Monolithic Application**:
   * Combine Orchestra, Upso, and HubSpot integration into one codebase (e.g., Django/Spring Boot).
3. **Cost-Effective Monitoring**:
   * Replace New Relic with AWS CloudWatch free tier.
   * Manual log analysis via Kibana.



## **6. Preventing Inconsistent Configurations**

### **Strategy 1: API Gateway Enforcement**

* Deploy a proxy layer (Kong/AWS API Gateway) to intercept HubSpot/Upso API calls.
* Block requests not originating from Orchestra (e.g., direct HubSpot UI changes).

### **Strategy 2: Immutable Configurations**

* Store configurations in Git (infrastructure-as-code).
* Require pull requests + CI/CD checks for changes.

### **Strategy 3: Real-Time Reconciliation**

* Use HubSpot/Upso webhooks to detect unauthorized changes.
* Trigger Orchestra to revert changes or alert admins.

## **7. Key Improvements & Security Considerations**

### **Improvements Over Initial Design**

1. **Orchestra as Single Source of Truth**: Enforced via technical controls (APIs/webhooks).
2. **Cloud-Native Scalability**: Kubernetes + serverless for multi-region redundancy.
3. **Automation Focus**: GitOps for configs, Terraform for infrastructure.

### **Security**

* **Zero-Trust Access**: OAuth2 for all services, VPCs for internal traffic.
* **Audit Logs**: Track configuration changes in PostgreSQL.
* **Encryption**: TLS for data in transit, AES-256 for data at rest.