Case Study

Three main components -

● Poolside assistant: IDE plugins that allow the extension of IDEs, like VSCode, to

natively utilize poolside capabilities;

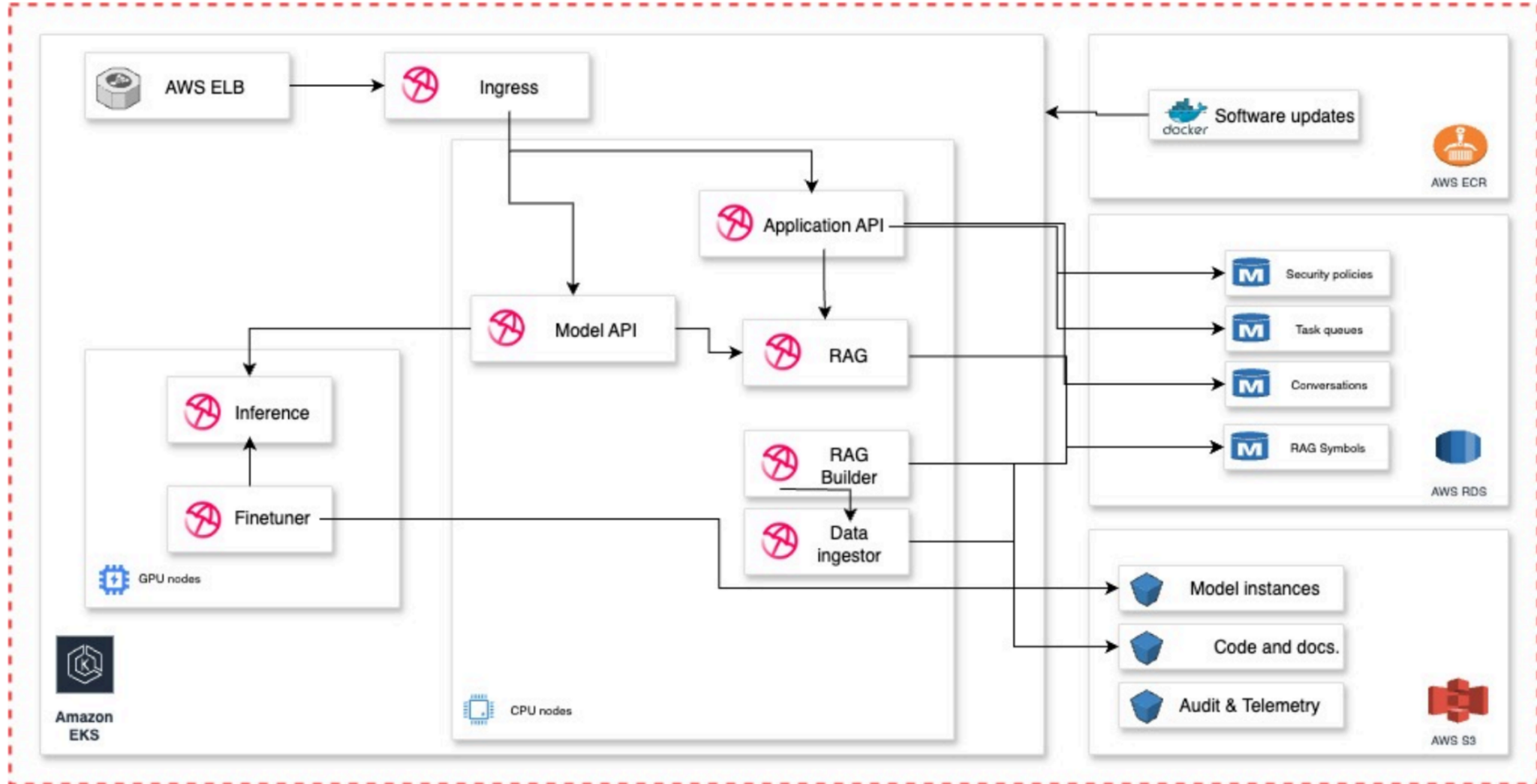
● poolside platform: a cloud service that can be installed in a customer VPC to

ensure data never leaves the customer premises;

● poolside model: a foundational model that supports all the underlying

capabilities that can accelerate our users' development process;

Reference Diagram



**Question 1:**

Promote a healthy customer engagement:

**a) Looking at this diagram, what would be your initial configuration concerns**

**and what information would you try to obtain from the customers?**

Some of my initial concerns, questions, and observations of the architecture diagram:

* Is this intended to be a Poolside provided diagram depicting the poolside architecture stack? Or, is this a customer provided document of their existing environment? For clarity, I’m seeking to understand if I’m looking if this a flawed architecture diagram from the provider viewpoint, or the customer viewpoint.
* Is this meant to be a “black box” solution - Poolside provides installation via helm charts, scripts, or other tooling, and the customer has no insights to the product?
* Who is expected to maintain, and update the platform, and on what cadence cycle?
* What is the demarcation point of support - e.g. considering each customer environment can be different, with different tooling, processes, and security, where is the limit for delivery and solutioning?
* Inference is an end service with no outputs, and does not feed into another service?
* Finetuner has outputs for Inference and Model Instances?
* I’d expect the RAG builder would feed into the RAG, and potentially the data-ingestor into the RAG builder?
* Are any services doing anything with Audit & Telemetry?
* Are there any additional services not depicted here, such as Secrets Manager, Certificate Manager, Kinesis, Elasticache for Redis?

Beyond the initial concerns, these are some observations and questions I have regarding core requirements:

1. **Network** -
   1. Is this all in the same VPC?
   2. Is PrivateLink, or cross-org access required? What accounts? What regions?
   3. ELB - Layer 4 or Layer 7? (NLB vs ALB)
   4. Presumably TLS + mTLS is required for client access
   5. Are there DNS Requirements for the Ingress
   6. Assuming this is an air-gapped environment, what is the mechanism for access? VPN?
   7. VPC CIDR Range + Subnets? Are RDS , GPU nodes, CPU nodes all in the same subnet? Different subnets?
   8. Security Groups + Port access - What service is talking to which service, in which directions.
   9. Network scope - are subnets sufficiently sized for scalability (nodes + pod IP space)
   10. Are Network Policies in place at the cluster level to limit and secure cluster traffic?
   11. What Ingress Solution is being used? Nginx, Ingress-Nginx, ALB, Istio?
   12. How are Images being built, tagged, pushed to ECR? Pull-through cache?
2. **IAM**
   1. What roles are required, with which permissions and for which services?
   2. Is IRSA + OIDC being used for EKS?
   3. Who has access to the environment and is maintaining it from an organizational perspective?
3. **Security**
   1. Is there sensitive data in the environment?
   2. Encryption at rest (RDS?)
   3. Encryption in Transit (TLS to the Load Balancer - mTLS through the cluster?)
   4. User Authentication - How are users authenticated / authorized? IAM? SCIM / SSO Integration?
   5. Are API Keys in use? Where are they generated? Are they stored, or hashed? Where is that kept? Expiration periods? Ability to revoke?
   6. How are Secrets kept / retrieved - Vault, Secrets Manager, CyberArk? SecretProviderClass in cluster?
4. **Compute**
   1. What size nodes are required to support? How many nodes are required?
   2. Can we auto-scale the GPU node pool ($$$)
   3. Are these shared clusters, or dedicated clusters (for Poolside)
   4. How much CPU, Memory does a “minimum” workload constitute?
   5. How is the application deployed - Helm Charts? Manifests? Flux/Kustomize/Argo?
   6. What is the nature of the workload - CPU intensive, Memory Intensive? Is it bursty, or consistent?
5. **Storage**
   1. How many requests per second?
   2. Support for concurrent read/write operations - Readers / Writer replicas
   3. Database sizing - how much space is required? How many connections are required? How much CPU / Memory is required?
   4. How is data ingested - assuming air-gapped environment, how are models and embeddings kept up to date?
   5. Schemas - Single RDS instance with multiple databases? Multiple instances?
   6. Regional vs Global - RDS vs Aurora support?
6. **Scalability/Reliability**
   1. Is there support for HA - Multi-regional, Multi-Zonal? (K8s nodes zonally distributed, regional ALB)
   2. Multi-database (replicas) support? Regional backups for cutover?
7. **Monitoring**
   1. Cloudwatch / S3?
   2. Event logging / SIEM Ingestion for security purposes?
   3. Is Audit Logging in place?
   4. Overall utilization, queries per second, data per query?
   5. Service failures - notifications and alerting
   6. OpenTelemetry / Prometheus endpoint support
8. **Maintenance and Operations**
   1. How are updates (images, runtimes, models) provided?
   2. How are CVE’s and security vulnerabilities kept up to date and mitigated?
   3. Can the system be kept up / running for updates (Blue/Green), or does it need to be taken down?
   4. How are schema updates handled (Alembic?)

**b) What would be your suggestion to ensure a healthy onboarding of a**

**customer with the platform?**

This depends quite largely on the current state and maturity of the organization and delivery model. The ideal state would be a low / no-code approach via AWS Marketplace Service offerings that can be installed into a VPC. This would present the core requirements necessary to configure the service, and create all the necessary configuration, roles, policies, necessary to deploy the offering. This can further be extended to a Terraform AWS module to do so as code, by providing customers with IaC if that is their preference. A fully fleshed out example module, with all variables populated and documentation detailing the options is ideal.

For a multi-service platform like this (RDS, S3, K8s, Load Balancer), are the primary customers you are interfacing with have all the appropriate permissions and credentials to implement? Ensure that we have the right customers, and resources engaged up-front, with the prerequisites necessary to facilitate smooth onboarding. Having a checklist of requirements, permissions, resources necessary can provide the customer with the opportunity to ensure they have everything ready.

Additionally, working through an MVP or PoC with the customer - walking through the configuration, deployment, and establishing the base / test case of functionality to ensure that the offering is working. Having a core customer champion is ideal, as you can “train the trainer”, and they can expand the knowledge and development internally.

Lastly, follow-through. The initial deployment and usage period has the most questions as it’s unfamiliar territory to users. As time goes by however, and they become accustomed to the service, they might ask more poignant, detailed questions that are more technical in nature. Ensure there is a technical resource in place for the customer who can quickly react and respond once in this phase.

**c) How would you ensure observability in case something happens in a**

**customer VPC?**

This is a tough question in an air-gapped solution, and has multiple considerations.

1. “If something happens” is wide-ranging - is this a service failure, a malfunction, a bad image, a failed schema migration? These manifest in different ways, and present different challenges to triage.
2. Is this a Poolside managed service? In the event something occurs in a customer VPC, is it expected that Poolside has the insight, telemetry, and monitoring to know that something occurred, or is it expected that a customer would be notified and raise the issue with Poolside to take action?
3. Expanding on point 2, assuming these are air-gapped, secure environments I would anticipate direct telemetry is not accessible. In scenarios like this, my initial direction would be that a diagnostic bundle can be exported to be provided either manually / with human intervention, or through a mutually secured file transfer such as s3. This could include telemetry data, configuration details, environment details, audit,event, access logs, container logs that can be used to understand the problem.
4. As this is intended to be an AWS backed and offered service, using and building and pre-packaging Cloudwatch Alerts / Alarms, SNS notifications and potentially Lambda’s for event driven triggers to generate and push a log bundle to perhaps an S3 bucket with an IAM policy that allows poolside to retrieve.
5. PrivateLink limited (via endpoint connection, restricted principals) telemetry for “call-home” data like a heartbeat.

**Question 2:**

2) Technical architecture and configuration decisions:

a) **Show us a simplified version of a deployment script where you create 2**

**containers that communicate with each other.   
The first service should have an API that returns a string.**

**The API needs to get the string from a second service that will get the string from a Postgres database.**

**The database is seeded with the “hello world” string.**

**We expect you to show us the system running end to end and to send us the scripts to bootstrap everything on our own (please provide any documentation you feel is**

**relevant).**

<https://github.com/chrisaboyd/Samples/tree/main/homework>

b) **Show us an architecture diagram of the system and explain the decisions**

**made. We will then ask questions about how you would evolve the**

**architecture to new requirements.**

<https://github.com/chrisaboyd/Samples/blob/main/homework/imgs/docker_compose_diagram.png>

c) **Imagine that now the original system presented in the diagram needs to**

**schedule and watch some background jobs. E.g. from time to time, the**

**model needs to be fine-tuned with new information. How would you**

**approach the problem and what technologies would you use?**

Model fine tuning can range between real-time / near-real-time to scheduled, cron’d, batched, or sync’d. Additionally, there is the possibility for ephemeral fine-tuning, such as context specific upserts (e.g. during a developer session, attaching a local code snippet).

As presented for scheduled / watched jobs, this could be achieved through:

* A landing zone such as S3 for storing the content
* (Optional for event based triggers such as high criticality upserts) - For cloud native, EventBridge, Lambda to notify a scheduler as new data is dropped into S3. An alternative solution could be just an EC2 instance with a bash script that watches for changes in the S3 bucket and calls for a scheduler
* A scheduler for the upsert, such as Airflow, Prefect, Control-M or just good old cron
* Compute for processing the upsert - K8s, ECS, AWS Batch to chunk and vectorize the data  
    
  Some additional considerations:
* How often does the fine-tuning occur? Is it often (minutely, hourly), or low frequency? This could mean triggered / on-demand upserts instead
* If this is real-time or near-real-time, a processing queue such as Kafka, Amazon Kinesis, or RabbitMQ)
* Amount of data to be chunked and upserted - will this usage and query performance?