

PoT - AWS

Method and Approach

Solution is building following IBM's e-business Reference Architecture (ebRA), Team Solution Design (TSD) Method, Architecture Description Standard 2 (ADS2) and AWS Well-Architected Framework for non-functional and operational aspects. Non-functional requirements, estimates and assumptions are based on Adidas e-commerce platform. Conversion rate is taken as 2% based on Shopify (1.75% benchmark) and industry reports. Each chapter of the report represents one work product that can be treated as separate document. TSD is presented with figure 1. Diagram se created with draw.io (<https://app.diagrams.net/>) that includes AWS pack with icons and reference diagrams.

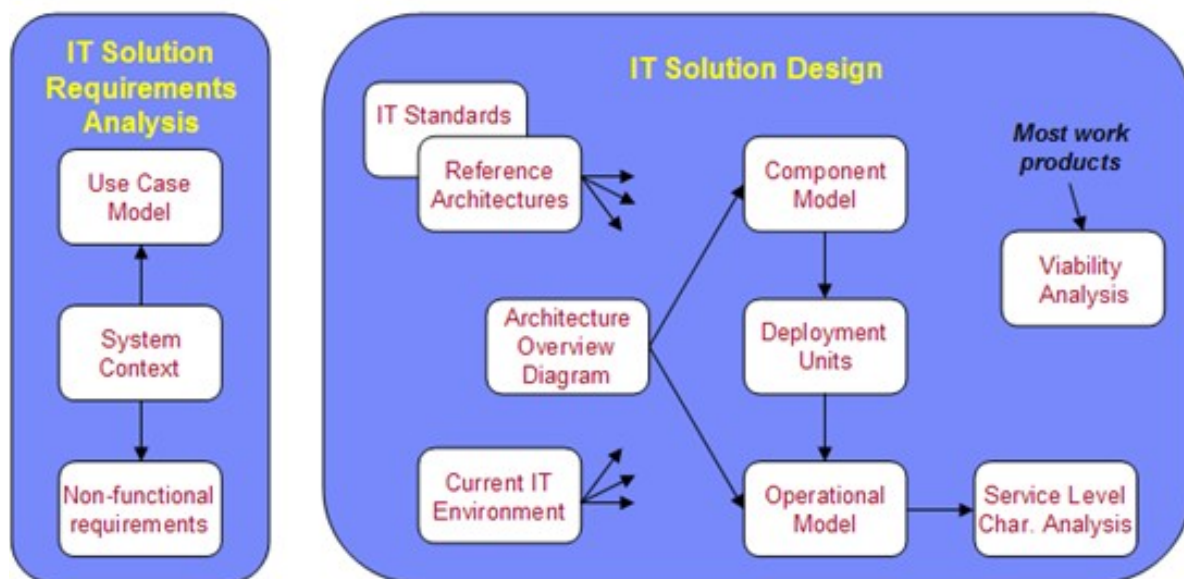


Figure 1 TSD simplified overview

Project description and general requirements

Technical Test Inc. is a luxury fashion house with stores all around the world. With the increased demand of e-commerce, the company has decided to open an online store.

Requirements

The main requirements for the project are:

Payments

The solution should accept two types of payments:

Online: payments made through the online store. These payments require validation by a third-party payment's provider (think Stripe, PayPal, etc.).

Offline: payments made in the physical store. These don't require validation as they are pre-validated on-site by the clerks (i.e. made in cash).

Performance

During peak shopping seasons the company expects a high volume of transactions, peaking at hundreds of thousands requests per second (most of them being read requests). The type of datastore/strategy chosen must be capable of handling such volumes without affecting the user experience. The biggest markets for the company are Europe and Asia. The solution must provide the **best service possible** for customers in both regions.

Analytics

Top executives of the company should be able to get insights about key metrics like sales, products' performance, customers, etc. The architecture should take this into account.

Observability

Support engineers should have a good view on how the system is performing, be able to identify key issues within the system (for instance failed payments), have events logs, etc.

Security / Privacy: The system contains payment data as well as sensible data from bcustomers. We need to ensure that all the data is well secured.

System context

The system context represents the entire system as a single object or process and identifies the interfaces between the system and external entities. Shown as a diagram, this representation defines the system and identifies the information and control flows that cross the system boundary.

The system context highlights important characteristics of the system: users, external systems, batch inputs and outputs, and external devices.

- External events to which the system must respond
- Events that the system generates that affect external entities
- Data that the system receives from the outside world and that must be processed in some way
- Data produced by the system and sent to the outside world

The purpose of the system context is:

- To clarify and confirm the environment in which the system has to operate.
- To provide the details at an adequate level to allow the creation of the relevant technical specification.
- Verify that the information flows between the solution to be installed and external entities are in agreement with any business process or context diagrams.

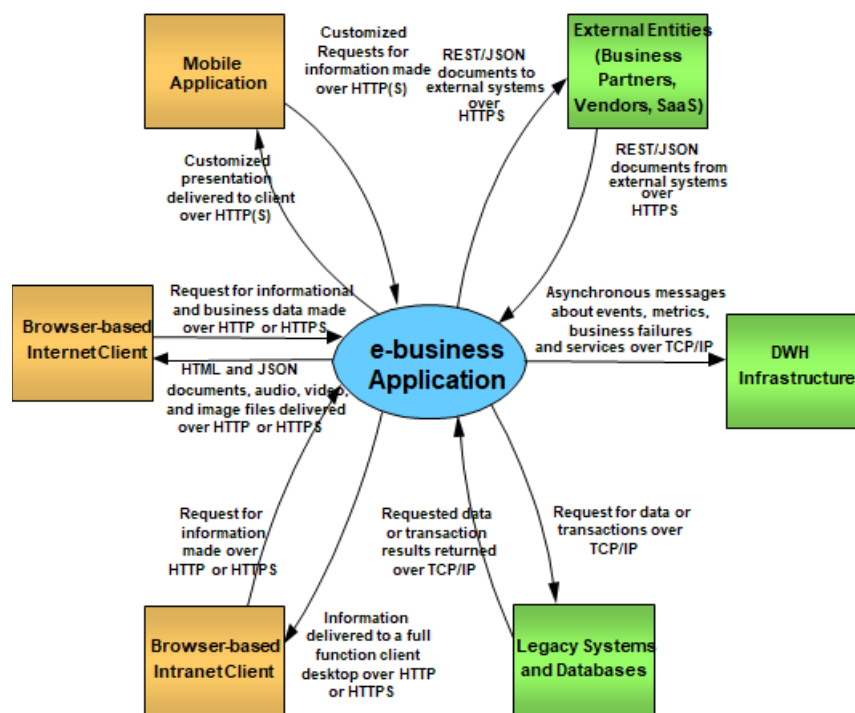


Figure 2 System Context Diagram

The context diagram shows the entire system represented as a single object or process, and identifies its interfaces with external entities of the system.

Mobile Application

Description	In-house developed or customized COTS applications (android, iOS). The benefit of having such an application is caching on the device and image resizing (on the server side). The application has all capabilities of Internet client but also ability to list items off-line and do application personalization and preferences. The application should be more about notifications and trends (content aggregator), in-app purchase is an option.
Type	System Actor
Number of users	High (assumption is over 1M)
Number of transactions	Medium (assumption is 1% when compared with Browser-based Internet client), 1000 at peaks
Frequency of transactions	Second (1000/second)
Volume of data	Low (assumption is under well below than 50KB per average transaction based on adidas.com), peak volume of about 0.5MB in the case of detailed view of high-resolution images.

Browser-based Internet client

Description	Part of the e-business Application that is deployed on the client side (e.g., React JS). It can be mobile device or computer (browser)
Type	System Actor
Number of users	Unknown (high and unpredictable), there are registered and unregistered users
Number of transactions	Unpredictable, over 100,000 at peaks (mostly read)
Frequency of transactions	Second (>100,000/second)
Volume of data	High (average 1.5MB for search and item details)

Browser-based Intranet client

Description	Part of the e-business Application that is deployed on the clerk machine (e.g., React JS). This is a different client with extra capabilities (e.g., POS) compared to the end-customer client.
Type	System Actor
Number of users	Low. Assumption is 500 (e.g. Adidas has 2500 stores world-wide). Users are clerks / Sales staff
Number of transactions	Low. Assumption is 100 per second (2 transactions per store, assuming 50 stores world-wide)
Frequency of transactions	100 per second (assuming clerk in peak hours does 2 per second and that they all click at the same time)
Volume of data	Medium (average transaction 0.5MB)

External entities

Description	External payment systems (Adyen, Paypal, Stripe, etc), external marketing systems, courier systems, etc. Ideally this should be minimized to few vendors to avoid complex integrations and increase security.
Type	System Actor(s)
Number of users	Low (assumption is 5-6 system users)
Number of transactions	At 2% conversion rate assumption is around 2000-4000 at peak

Frequency of transactions	Per second, 2000-4000 at peaks
Volume of data	Low (<50KB per transaction)

DWH Infrastructure

Description	External DWH infrastructure (it can be cloud based like Amazon Glue or some legacy corporate system based on Oracle, IBM, etc.). It is asynchronous communication in one direction. Managers do not need to see real-time data, near real-time is sufficient. It holds required subsystems (ETL, DWH, Data marts and reporting engine)
Type	System Actor
Number of users	1 (DWH system actor)
Number of transactions	Low (defined by triggering mechanism, assuming scheduler, however it can be defined on-demand) - assuming 1
Frequency of transactions	1 per minute
Volume of data	High (0.5GB per minute) assuming only customer data, time, status and service invoked.

Legacy systems

Description	Those include ERP, Warehouse system, potentially MDM, etc. The main challenge is to define where is master data for items (warehouse stock quantities, locations, etc). It can be in the e-business application, or it can be stored in external system. In the case of true distributed e-business application the source is inside application and systems are just notified with updates just like in the DWH case. Other option is more complex process based that is based on request response and requires request/response communication where those other systems would be seen as performance bottleneck.
Type	System Actor
Number of users	Assumption is 1 (e.g. SAP system user)
Number of transactions	At 2% conversion rate assumption is around 2000-4000 at peak
Frequency of transactions	Per second, 2000-4000 at peaks
Volume of data	Low (<10KB per transaction)

Non-Functional Requirements (NFRs)

The non-functional requirements specify the service levels the system must satisfy, the required non run-time properties of the system, and the constraints to which the system must conform. Non-functional requirements may apply to the system as a whole, to parts of the system, or to particular use cases.

Non-functional requirements consist of the following.

Service level requirements (SLRs), define run-time properties that the system must satisfy. SLRs include:

- capacity and performance (volumetrics)
- availability
- scalability
- security
- system management

The purpose of the non-functional requirements document is:

- To specify required properties of the system.

- To define constraints on the system.
- To enable early system sizing and estimates of cost.
- To assess the viability of the proposed system.
- To give a basis for designing the operational models.
- To provide input to the component design

Capacity and Performance (Volumetrics)

Throughput for read only services - end customer

Throughput should be estimated to support peak hours during specific periods (e.g., Holidays, marketing promotions) for several hundred thousand request per second. It can be unpredictable. Average request/response is about 1.5 MB. Due the low number of transactions from the Clerk side and mobile application, this can be all neglected and included in general Capacity and Performance.

General	e-business application should support high throughput reading capabilities and scale accordingly.
Performance and Capacity	Average load is estimated at 10000 transactions per second. Peak: >100k transactions per second. Loading time less than 2 sec.
Latency	<100ms, Note: AWS does support that in 99 percentiles through: Asia Pacific (Singapore) <i>ap-southeast-1</i> EU (Frankfurt) <i>eu-central-1</i> Assumed published findings. Latency plays a huge part in customer's perception of a high-quality experience and was proved to impact the behaviour of users to some noticeable extent: with lower latency generating more user engagements. This observation has also been confirmed several times by few large companies: Amazon: 100 ms of extra load time caused a 1% drop in sales (Greg Linden). Google: 500 ms of extra load time caused 20% fewer searches (Marissa Mayer). Yahoo!: 400 ms of extra load time caused a 5-9% increase in the number of people who clicked "back" before the page even loaded (Nicole Sullivan).
Availability	99,99%, 24x7x365 (supported by AWS Fargate)

Throughput for write services - end customer

Assumption is that number of transaction is 8% of reads (2% conversion rate and 4% for metrics, login, adding/deleting items from shopping cart, etc)

General	e-business application should perform medium throughput far less then read services.
Performance and Capacity	Average load is assumed to be 1000 transactions per second. Peak is assumed at 6000 transactions per second; however, it can be unpredictable. Response time less than 1 sec.
Latency	<100ms, Note: AWS does support that in 99 percentiles through: Asia Pacific (Singapore) <i>ap-southeast-1</i> EU (Frankfurt) <i>eu-central-1</i> Assumed published findings. Latency plays a huge part in customer's perception of a high-quality experience and was proved to impact the behaviour of users to some noticeable extent: with lower latency generating more user engagements.

	<p>This observation has also been confirmed several times by few large companies:</p> <p>Amazon: 100 ms of extra load time caused a 1% drop in sales (Greg Linden).</p> <p>Google: 500 ms of extra load time caused 20% fewer searches (Marissa Mayer).</p> <p>Yahoo!: 400 ms of extra load time caused a 5-9% increase in the number of people who clicked “back” before the page even loaded (Nicole Sullivan).</p>
Availability	99,95%, 24x7x365 (supported by AWS Fargate and AWS Lambda)

Throughput for write services - Payment processing

General	e-business application should perform medium throughput at about 1/3 of total write services.
Performance and Capacity	Average load is assumed to be 500 transactions per second. Peak is assumed at 2000-3000 transactions per second; however, it can be unpredictable. For payment processing response time is up to 5 seconds based on available resources.
Latency	<1 second Mostly depends on the payment processor
Availability	99,95%, 24x7x365 (AWS Lambda)

Availability

Inherited through cloud infrastructure (minimum 99,95% for writes, 99,99% for reads – as defined by Amazon SLRs). General requirement is 99,99 % 365x7x24. Only 99,95 is guaranteed by AWS Lambdas (writes). 3rd party applications such as marketing tools and external systems (ERP, DWH) are not considered for this requirement due asynchronous nature of the communications (messaging or triggering the e-business application). Clerk part of application should be availability of 99.95% during regular business hours in each region.

Backup & Recovery

Multi-region (multi-site) active-active (RPO near zero, RTO potentially zero) by utilizing cloud infrastructure.

Scalability

By utilizing Cloud infrastructure with automation (auto-scale, and lambdas)

Security

Based on AWS Well-defined framework, the security pillar describes how to take advantage of cloud technologies to protect data, systems, and assets in a way that can improve security posture.

Implement a strong identity foundation: Implement the principle of least privilege and enforce separation of duties with appropriate authorization for each interaction with AWS resources. Centralized identity management, and aim to eliminate reliance on long-term static credentials.

Enable traceability: Monitor, alert, and audit actions and changes to your environment in real time. Integrated log and metric collection with systems to automatically investigate and take action.

Apply security at all layers: Apply a defence in depth approach with multiple security controls. Apply to all layers (for example, edge of network, VPC, load balancing, every instance and compute service, operating system, application, and code).

Automate security best practices: Automated software-based security mechanisms improve ability to securely scale more rapidly and cost-effectively. Create secure architectures, including the implementation of controls that are defined and managed as code in version-controlled templates.

Protect data in transit and at rest: Classify your data into sensitivity levels and use mechanisms, such as encryption, tokenization, and access control where appropriate.

Keep people away from data: Use mechanisms and tools to reduce or eliminate the need for direct access or manual processing of data. This reduces the risk of mishandling or modification and human error when handling sensitive data.

Prepare for security events: Prepare for an incident by having incident management and investigation policy and processes that align to your organizational requirements. Run incident response simulations and use tools with automation to increase your speed for detection, investigation, and recovery.

Use Case Model

The Use Case Model work product describes the functional requirements of the system under development. The model uses structured text to specify the ways in which users in specific roles will use the system (i.e., use cases). The textual descriptions describing the use cases are from a user's point of view; they do not describe how the system works internally or its internal structure or mechanisms.

The Use Case Model consists of the following:

- Preamble - general considerations, style and assumptions
- Actors - their role and brief description
- Use case list - the names and numbers of the use cases
- Use cases - definitions of each kind of system usage using a standard template

The main purpose of use case modelling is to define the boundary of the proposed software in terms of its functional behaviour. It is the part of the "contract" between the client and the development team: it tells the client what will be delivered, and it tells the development team what it must build. Other purposes are listed below:

- Provides a basis of communication between end users and system developers
- Is the primary driver for estimating the application development effort
- Provides a basis for planning the development of the releases
- Provides a basis for identifying objects, object functionality, interaction, and interfaces
- Provides the primary basis for defining the user interface requirements
- Provides a basis for defining test cases
- Serves as the basis for acceptance testing
- Provides a basis for producing user support materials, such as user documentation and electronic performance support interventions

Simplified use case is given with Figure 3.

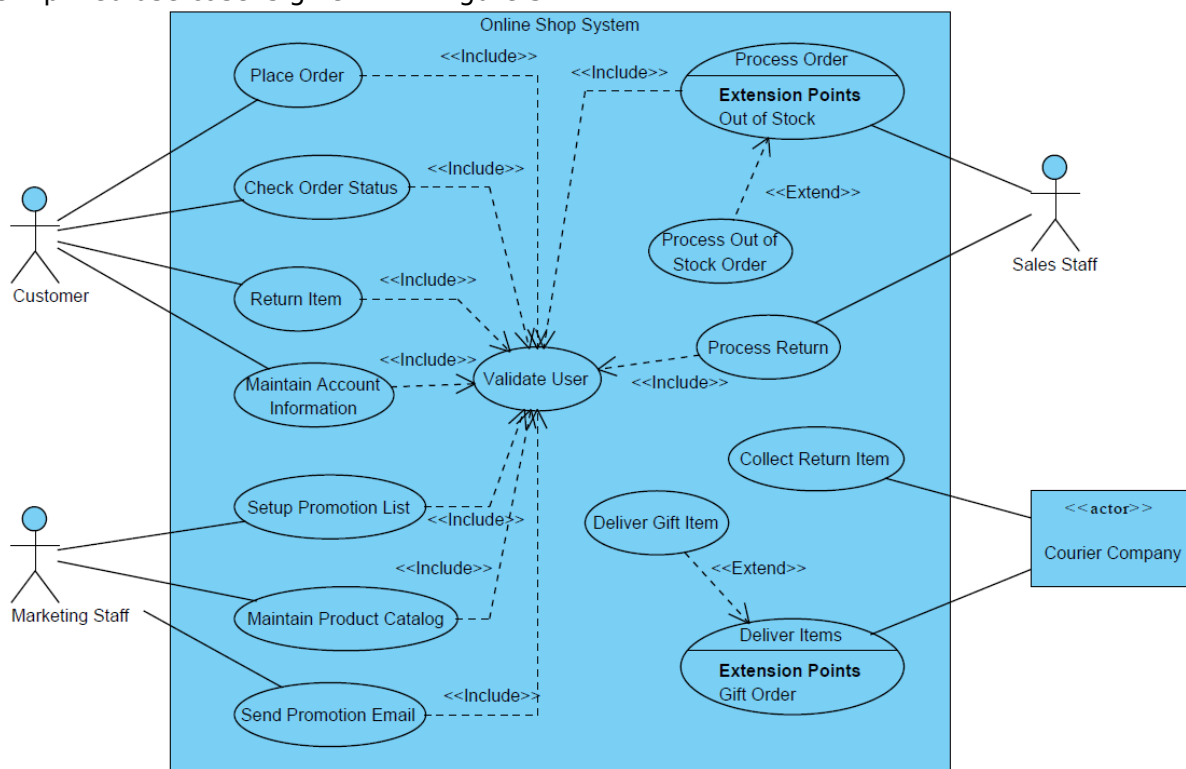


Figure 3 Use case diagram

Normal scenario (Customer)

1. Customer enter login information (optional)
2. System display product menu / search results
3. Customer add items to shopping cart
4. System display message indicate the item added to shopping cart
5. Customer proceed to checkout
6. System ask user provide shipping and billing information
7. Customer provide shipping and billing information
8. System confirm the shipping information, process the order and ship out the items
9. Customer receive the items

Normal scenario (Sales staff)

1. Clerk enters login information (optional, if not logged-in)
2. System display product menu based on Item number (scanner, manual entry, etc.)
3. Clerk add items to shopping cart
4. System display message indicate the item added to shopping cart
5. Clerk process the payment (POS)
6. Customer receive the items

Normal Scenario (currier)

1. System issue delivery request to courier company
2. Courier company collect items from warehouse, pack it and ship it out
3. Courier company mark delivery complete
4. System confirm order close

Fail to Deliver Items(currier)

1. System issue delivery request to courier company
2. Courier company collect items from warehouse, pack it and ship it out
3. Courier company mark delivery fail since no one accepting the package
4. System mark delivery fail, notify sales staff contact customer to reschedule deliver

Architectural Decisions

Architectural Decisions documents important decisions about any aspect of the architecture including the structure of the system, the provision and allocation of function, the contextual fitness of the system and adherence to standards.

An architecture is understood partly through the record of the important decisions made during its development. The justification and evaluation criteria are recorded alongside the decision or by reference to more generally applicable principles, policies and guidelines, which are found in other work products or in external references.

The purpose of the Architectural Decisions work product is to:

- Provide a single place to find important architectural decisions
- Make explicit the rationale and justification of Architectural Decisions
- Preserve design integrity in the provision of functionality and its allocation to system components
- Ensure that the architecture is extensible and can support an evolving system
- Provide a reference of documented decisions for new people who join the project
- Avoid unnecessary reconsideration of the same issues

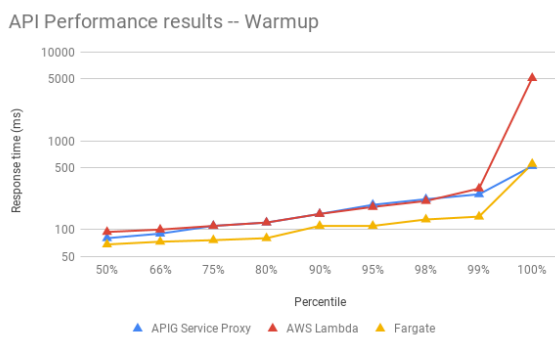
Decision Descriptions

Subject Area	Infrastructure	Topic	Cloud
Design Decision	AWS shall be used as a target deployment platform together with Global Accelerator, Cloudfront and Route 53. HYBRID	Id.	D1

Issue or Problem Statement	Multi-regional setup is required with high availability, extreme and unpredictable load, active-active topology is required.
Assumptions	Price is reasonable compared to alternatives
Motivation	Customer preferences and current infrastructure
Alternatives	Azure, GCP, IBM Cloud, AWS (Route 53, instead of accelerator)
Decision	HYBRID. Deployment on AWS with Global Accelerator, Cloudfront and Route 53
Justification	Skilled team, preferred platform, according with NFRs. Global Accelerator provides: Improved latency, packet loss, & overall quality. Avoid network interconnect capacity conflicts. Greater operational control. Realtime recovery and DynamoDB global tables.
Implications	Potential lock-in to AWS, price of the Global Accelerator

Subject Area	Database	Topic	RDBMS vs NoSQL
Design Decision	Amazon DynamoDB (NoSQL) is going to be used together with global tables	Id.	D2
Issue or Problem Statement	In order to support multiple regions and Active-active configuration across regions, fully managed, multi-region, and multi-active database that delivers fast, local, read and write performance for massively scaled, global applications are required. Global tables replicate DynamoDB tables automatically across the choice of AWS Regions.		
Assumptions	Microservice architecture		
Motivation	NFRs, performance, management, ease of implementation		
Alternatives	Aurora RDBMS		
Decision	Amazon DynamoDB with global tables for write operations		
Justification	Easily configurable and fast enables read and write, faster than Aurora RDBMS when it comes to write forwarding and transaction. DynamoDB is a fully managed, multi-region, and multi-master database, allowing for the first time to build globally distributed applications.		
Implications	Must be microservice architecture (event-sourcing, saga patterns, etc), no 2-phase commit support, possibility of data duplication.		

Subject Area	Architecture style, programming languages	Topic	Containers / Functions and implementation languages
Design Decision	Serverless. Hybrid. Containers for reads (AWS Fargate with autoscaling), Lambdas for writes. Lambdas implemented in Python. Containers operate Java with Spring boot.	Id.	D3
Issue or Problem Statement	System must implement fast reads, small percentage of the transactions are write transactions. Pricing should be considered.		
Assumptions	Number of reads is much higher (over 90%) of the traffic, writes can be performed asynchronously (e.g. logging events) small percentage of writes require synchronous transactions (e.g. payment processing, checkouts and stock upgrades). Most of the application is based on Docker images. Team has both skills. Performance when the shopping is "over" is not that relevant (user can wait for few seconds to get confirmation).		

Motivation	Containers (Fargate performs faster in high loads especially for tasks such as reads and warm-up time is significantly lower). For Lambdas warm-up time for Python is up to 400x faster (reports show that Python <2ms, while Java >800ms). Available at: https://medium.com/the-theam-journey/benchmarking-aws-lambda-runtimes-in-2019-part-i-b1ee459a293d																																								
Alternatives	All containers. All lambdas. All Java implementation. All Python																																								
Decision	Microservices, combination of Containers (autoscaling) based on Java for reads and Python based Lambdas for writes.																																								
Justification	<p>Java with Fargate outperforms Python, while Python implementation of Lambda functions have the fastest response time and latency. During checkout due integration with</p> <p>API Performance results – Warmup</p>  <table border="1"><caption>Approximate data points from the API Performance graph</caption><thead><tr><th>Percentile</th><th>APIG Service Proxy (ms)</th><th>AWS Lambda (ms)</th><th>Fargate (ms)</th></tr></thead><tbody><tr><td>50%</td><td>~100</td><td>~100</td><td>~80</td></tr><tr><td>66%</td><td>~120</td><td>~120</td><td>~100</td></tr><tr><td>75%</td><td>~150</td><td>~150</td><td>~120</td></tr><tr><td>80%</td><td>~180</td><td>~180</td><td>~140</td></tr><tr><td>90%</td><td>~250</td><td>~250</td><td>~180</td></tr><tr><td>95%</td><td>~300</td><td>~300</td><td>~220</td></tr><tr><td>98%</td><td>~350</td><td>~350</td><td>~280</td></tr><tr><td>99%</td><td>~400</td><td>~400</td><td>~350</td></tr><tr><td>100%</td><td>~500</td><td>~8000</td><td>~450</td></tr></tbody></table>	Percentile	APIG Service Proxy (ms)	AWS Lambda (ms)	Fargate (ms)	50%	~100	~100	~80	66%	~120	~120	~100	75%	~150	~150	~120	80%	~180	~180	~140	90%	~250	~250	~180	95%	~300	~300	~220	98%	~350	~350	~280	99%	~400	~400	~350	100%	~500	~8000	~450
Percentile	APIG Service Proxy (ms)	AWS Lambda (ms)	Fargate (ms)																																						
50%	~100	~100	~80																																						
66%	~120	~120	~100																																						
75%	~150	~150	~120																																						
80%	~180	~180	~140																																						
90%	~250	~250	~180																																						
95%	~300	~300	~220																																						
98%	~350	~350	~280																																						
99%	~400	~400	~350																																						
100%	~500	~8000	~450																																						
Implications	Potential vendor lock-in (Lambdas), separate development teams (Python for Lambdas and Java for fast searches and reads).																																								

Application Overview Diagram

The architecture overview diagram represents the governing ideas and candidate building blocks of IT system and enterprise architecture. It provides an overview of the main conceptual elements and relationships in architecture, including candidate subsystems, components, nodes, connections, data stores, users and external systems. Typical is given by the existing e-business pattern (Reference architecture)

The main purpose of the architecture overview diagram is to communicate a simple, brief, clear and understandable overview of the target IT system.

This type of diagram is produced at several levels, which include:

- Enterprise level
- Services level
- IT Services View

At an enterprise level, an Architecture Overview Diagram is often produced as part of an overall IT Strategy.

The purpose of the architecture overview diagram is to:

- Communicate a conceptual understanding of the target IT system
- Provide a high-level shared vision of the architecture and scope of the proposed IT system
- Explore and evaluate alternative architectural options
- Enable early recognition and validation of the implications of the architectural approach
- Facilitate effective communication between different communities of stakeholders and developers
- Facilitate orientation for new people who join the project

Enterprise View

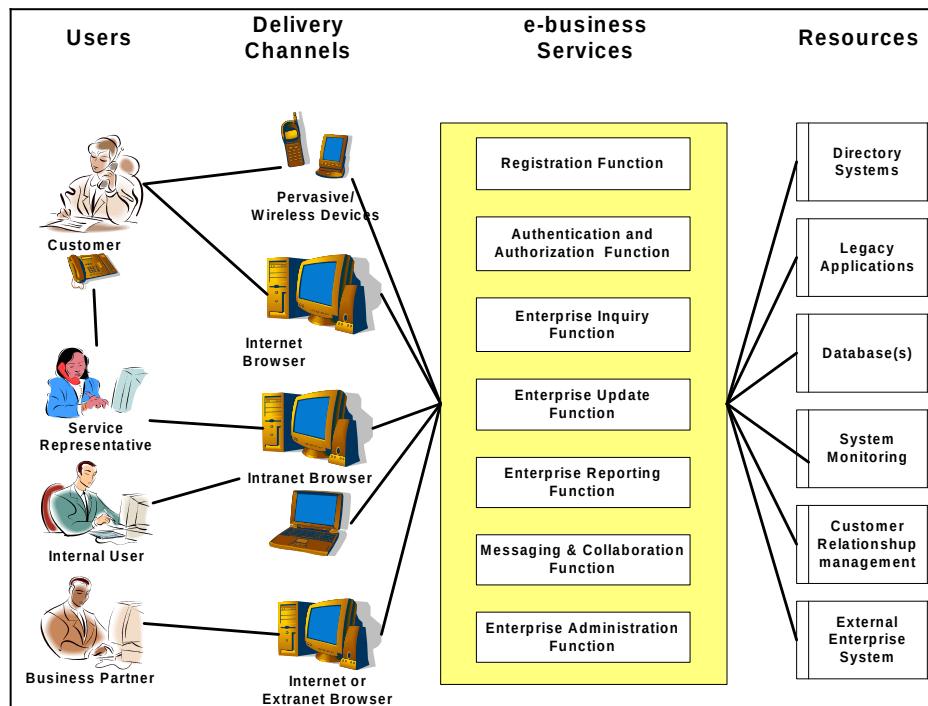


Figure 4 Enterprise View

Users and delivery channels

The Enterprise View depicted above will allow different types of audiences to access the on-line store over various delivery channels:

- Service Representatives users and other authorised personnel will access the applications using desktop client (POS) or Web browsers.
- Different users would see different interfaces to the system; while some users will be able to view or act upon shopping and payments, others will have only access to summaries, statistics, analytics and other reports.
- Data is offered in standard formats:
 - UTF-8 for representation of characters (letters, digits and other signs), as this scheme provides ability to correctly represent all characters in different languages and foreign names that might appear in delivery addresses (e.g. Important for Asian markets)
 - PDF for printing-ready copies of documents (invoices, etc) that can be preserved or printed according to user's needs

Each channel provides the user with access to certain functions of the e-business applications. The capabilities that are made available and the presentation style may differ depending on the type of user and the channel.

e-business Services

The following types of business functions will be supported by the environment:

- Authentication and Authorization – This function will authenticate the users using some form of challenge to authenticate the user (determine users' identity) and will validate if users are authorized to access the requested resources. Access to all other applications and systems is always governed by this component.
- E-business application functions perform core services in this engagement, providing automated and integrated on-line store.
- Enterprise Services – These services provide common and reusable business-related capabilities to the applications (implemented as micro-services).
- Integration – This function is providing formal and consistent interface for some identified components of application which will require integration with external systems (payments).

Resources

Resources are accessed by the applications to perform the functionalities supported by the Application Services Tier:

- **Directory Server** – contains information about all persons authorised to use the system. Only information relevant to use of the system is kept here, like user's full name, a credential needed to prove user's identity (such as encrypted password), role of a user, and addressing information used to notify the user about relevant system events.
Roles are used to raise the quality of system administration – as groups of users share same or similar needs and authorisations regarding system use, instead of defining same attributes to every user in such group, a role with attributes is defined and applied to all users in the group. Administration is simpler and less prone to errors, and also it is easier to check if users are given correct authorities.
- **Authorization Policy Server** – stores policies (sets of rules) that describe conditions needed to grant access to any part of the system. For example, a condition can define that particular user role has the privilege to access particular system part and do a specific action. These privileges are further mapped to the policies set by the application owners.
- **Databases** – used to store the application data and managed content. Cloud based NoSQL with automatic on-line maintenance capabilities.
- **Legacy Applications and Services** – can be accessed through the Integration capabilities to perform/execute applications or services running on legacy systems if available (DWH, ERP, CRM).
- **External Applications** – system is planned to be deployed and used for long time, so access to different external systems is likely to happen (payment processing, courier, etc)

Services View

Presents services provided by specific tier, organised vertically and horizontally, depending on selected platform vertical services can be provided by infrastructure (case of PaaS).

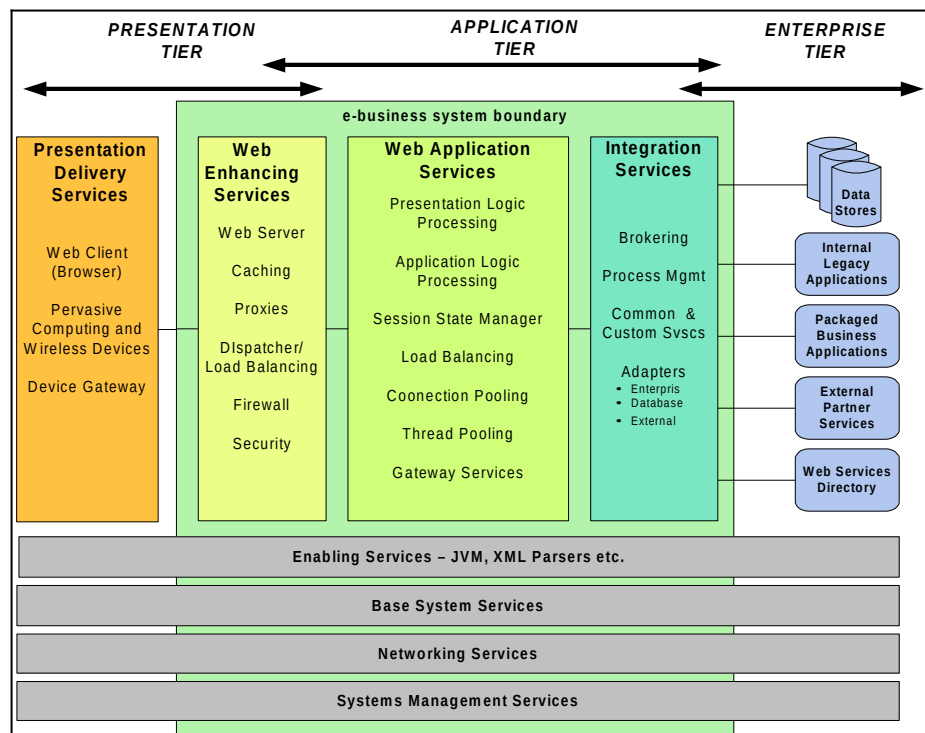


Figure 5 Services View

The Presentation Services, with its associated technologies, support the presentation of application information to end-users. The overall objective is to do this in an intuitive fashion, allowing the end user to interact with the system in a very natural, consistent

manner. An additional role of the Presentation Delivery Services component is to unify and draw from the technology and power of other resource managers and to abstract away their individual complexities from the user.

Delivery Enhancing Services include components that help enhance the architecture and allow it to provide better service levels for key Non-functional requirements such as performance, scalability, availability and security. These components include:

- Cache Managers that help cache commonly used objects and thereby improve response time and reduce the load on other architectural components. As caches will be geographically dislocated, caching shall be done only on selected data to make sure that caching does not result in creating business risks (such as unauthorized access to sensitive data, loss of data, or inconsistencies among all copies of data).
- Load balancers that route traffic to multiple redundant servers.

Integration Services include access to existing application, systems, and data sources, both internal and external to the enterprise as well as access to new common services shared and reused by all applications. This layer isolates the complexities of integration from the application services components. Integration and Enterprise Services components include the set of functions required by individual components to communicate with one another in a distributed computing environment.

The Authentication and Authorization Services provide the mechanisms to challenge and authenticate users as well as to verify their right to access the requested resources.

Management Services (or Management Information Services, MIS) are services acting upon non-operational data and in non-transactional manner. Examples are reports, statistics, analytics and similar.

Manageability of the platform resources and monitoring security related events is a critical requirement in the environment. Services must be provided to satisfy the following needs:

- Configuration information for resources should minimize administrator involvement.
- Basic error logging and fault isolation functions are required to support manual and automated problem determination and resolution.
- Common standard management definitions and protocols should provide for the collection of error and performance data and the application of changes and fixes.
- Monitoring and reporting response time and performance data
- Monitoring and reporting availability data
- Providing data confidentiality during transmission and storage, where required

Simplified IT Systems View

This view represents high level of systems that serves as Logical operational model where individual systems are mapped on AWS and it is base line for deploying data, execution and presentation units on Cloud provided infrastructure and services (e.g. static content goes to CDN and S3)

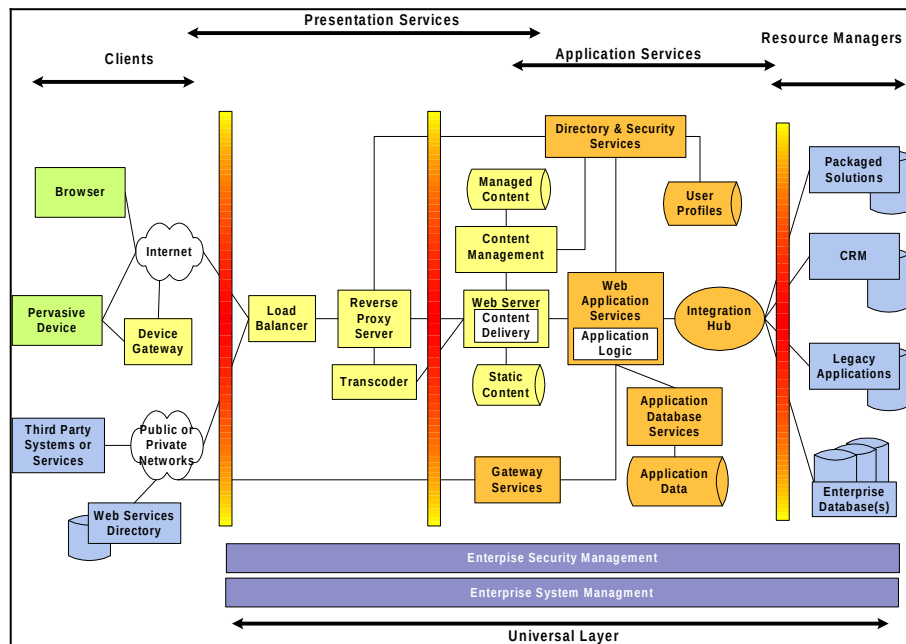


Figure 6 IT System View

Component model

The Component Model work product describes the entire hierarchy of components in terms of their responsibilities, their interfaces, their (static) relationships, and the way they collaborate to deliver required functionality. A component is a relatively independent part of a system. It is characterized by its responsibilities and by the interface(s) it offers. Components are use case realizations (subsystems) that encapsulate business functionalities. Component realization is via microservices, meaning one logical component is forged by multiple microservices, each service can have its own datastore and separate container.

Main components of the system are:

- Security subsystem (user management, identity management, AAA)
- Logging subsystem (events)
- Shopping subsystem (searches, lists, item details)
- Item management (updates) -> to be implemented as Lambda
- Order management subsystem (with shopping cart operations) -> to be implemented as Lambda
 - Utilizes Global Accelerator
- Integration subsystem (communication with external entities) -> to be implemented as Lambda
- Payment component -> to be implemented as Lambda

Deployment units

Data units such as static data (product descriptions, photos) are to be placed on S3 and cached through CDN

Transactional data (orders, events, coloration numbers) are to be placed directly in DynamoDB, no caching

Execution units for read operations are located on Fargate (containers), client (React.js, mobile application) and external systems (DWH/BI infrastructure). Presentation units are on the client side and rendered by execution units from the client (JavaScript rendering HTML, mobile application build in)

Logical operational model

On top there is Global Accelerator that routes traffic to appropriate region. Global tables are used only for writing transactional data (e.g. orders). Global Accelerator enables this design when multi-master database is required. It is quite easy to setup and supports multiregional disaster recovery.

Downsize is cost and degradation of response time since global accelerator can be configured to two IP addresses and works globally, unlike Route 53 with CloudFront that caches to the nearest location for reads.

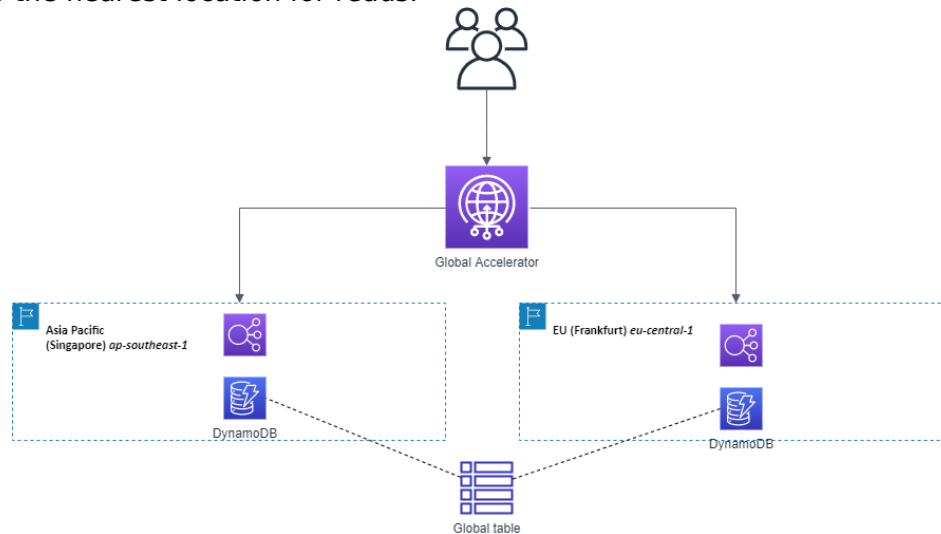


Figure 7 Global accelerator and Global table

Therefore, hybrid approach is used based on Route 53 simple routing policy as shown in figures bellow. It should be combined with CloudFront. Basically, only write operations go to Global accelerator and that can be managed also through application logic since application is based on microservices and each microservice can have its own database for its purposes.

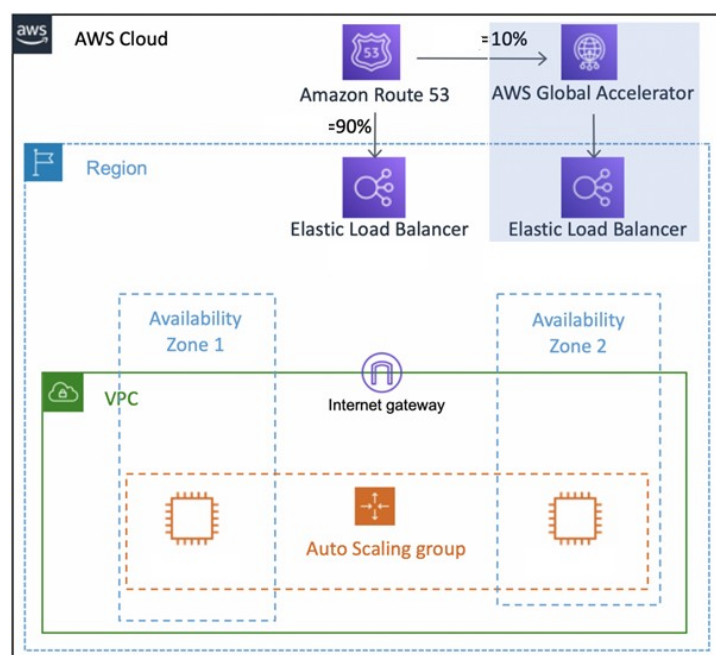


Figure 7 Hybrid routing (10% of transactions go through GA for writing)

Autoscaling group is defined per region for availability zones. Fargate handles policy as shown with Figure 8. Lambdas scale by nature with ramp time. AWS GA and Amazon Route 53 have different IP addresses and domains, since client can have multiple open connections.

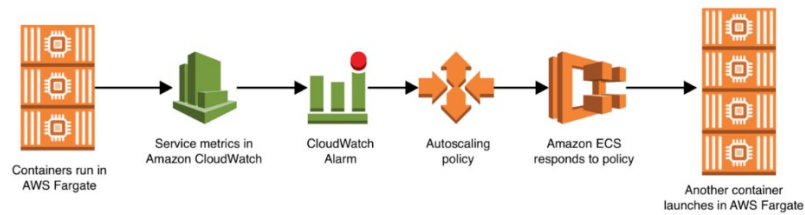


Figure 8 High level view for region with Fargate autoscaling concept

Data Warehousing

Assumption is that client already has some DWH solution and therefore is put out of the context. Lambdas are triggered in order to perform the ETL jobs and fill DWH as shown below.

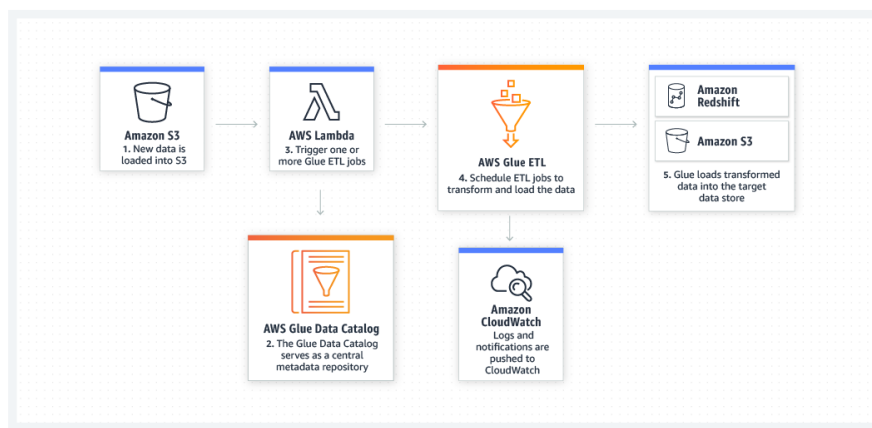


Figure 9 Integration with the DWH

Other option is AWS Glue Elastic Views that enable usage of SQL to create materialized views. Use these views to access and combine data from multiple source data stores, and keep that combined data up-to-date and accessible from a target data store. The AWS Glue Elastic Views preview currently supports Amazon DynamoDB as a source which is aligned with given architecture.



Figure 10 Integration with the DWH (Glue Elastic Views)

Development and testing environment.

Should follow standard CI/CD scenario as described by amazon. From Source stage (S3/AWS Code commit) to deployment stage test that has Code Deploy for containers to the Code Deploy for the production.