USING THE C4 MODEL TO DOCUMENT **“BOOKSTORE SYSTEM”**

Level 1 – Context Diagram for Bookstore System

The system allows users to interact with book records. This diagram will *require* to contain *four details* we need for our stakeholders understanding about the context:

- There will be three different types of users: *Public User*, *Authorized User,* and *Internal User*.
- It depends on an external *Identity Provider System* for authorization purposes.
- It collects the published book details from an external *Publisher System*.
- It connects with a 3rd party Shipping Service to handle the book delivery.

Level 2 – Container Diagram for Bookstore System

Zooming in, this diagram will mention more on technical details about the actual system, it also explicitly indicates the technologies we use on the concrete containers.

Front-store Application:

- Provide all the bookstore functionalities to both public and authorized users.
- Developed with JavaScript & ReactJS.
- Interact with Public Web API and Search API to search book and place order.

Back-office Application:

- Provide all the bookstore administration functionalities to internal users.
- Developed with JavaScript & ReactJS.
- Interact with Admin Web API to administrate books and purchases.

Search API:

- Developed with Go.
- Allows ONLY *authorized users* to search books information using HTTPs. The user requests will be authorized by the external *Identity Provider System.*
- Connected with a Search Database to get the book information.

Search Database:

- Use *Elasticsearch* to store the book searchable data.
- The search data will be produced by *Book Event Consumer*.

Public Web API:

- Developed with Go.
- Allows publish users to search books information using HTTPs.
- Connected with *Bookstore Database* to read/write the data.

Admin Web API:

- Developed with Go.
- Authorized by external *Identity Provider System*.
- Allow ONLY *internal users* to manage books and purchases information using HTTPs.
- Connected with *Bookstore Database* to read/write the data.
- Publish book events to *Book Event System*.

Bookstore Database:

- Use PostgreSQL to store the data.

Book Event System:

- Using Apache Kafka 3.0.
- Handle the book published event and forward to the Book Event Consumer.

Book Search Event Consumer:

- Developed with Go.
- Handle book update events and write to Search Database.

Publisher Recurrent Updater:

- Listen to external events from Publisher System.
- Developed with Go.
- Use Admin Web API to update the data changes.

Level 3 – Component Diagram for Admin Web API

Zooming into the Admin Web API, it contains set of components described on below:

- A service which name is *Book Service*, allow administrating book details. It reads and writes data to Bookstore Database.
- A service which name is *Authorizer*. This service will authorize the internal users using external Identity Provider System.
- There is a service named *Book Events Publisher* to publish book-related events to the *Book Event System*.

Deployment Diagram – Production Deployment on AWS (optional)

The system will be deployed using AWS EKS. The architecture will be provisioned with the below information:

- The target region is Singapore (***ap-southeast-1***).

- ***Amazon Route 53*** will be used to route the DNS request to an ***Application Load Balancer (ALB)***.
- The Front-store Application will be deployed to ***S3*** in the same region along with CloudFront distribution to forward the request.
- The EKS worker nodes will be deployed in the Private Subnets: ***private-net-a***, ***private-net-b***:
    - An EC2 instance named ec2-a launches in private-net-a.
    - An EC2 instance named ec2-b launches in private-net-b.
- ***Backoffice Application, Search API***, ***Admin Web API***, ***Public Web API*** will be deployed on ec2-a as microservices.
- Bookstore Database is deployed using ***PostgreSQL RDS*** service in private-net-b.
- Search Database is deployed using ***AWS OpenSearch*** in private-net-b.
- ***Book Event Consumer, Book Event System*** will be deployed on ec2-b as microservices.
- Identity Provider System and Publisher System are external systems. So, we will not document it. Their workloads are considered in AWS cloud as well.

Dynamic Diagram – Workflow CI/CD using AWS services (optional)

The workflow includes following steps:

1. **Developer** commits and pushes changes to a source code repository.
2. A webhook on the code repository triggers a **CodePipeline** to build in AWS.
3. **CodePipeline** downloads the source code and starts build process.
4. **CodeBuild** downloads the necessary source packages and start running commands to build and tag a local docker image.
5. **CodeBuild** pushes the container image to **Amazon ECR**. The container image is tagged with a unique label derived from the repository commit hash.
6. **CodePipeline** deploys image on **Amazon EKS**.