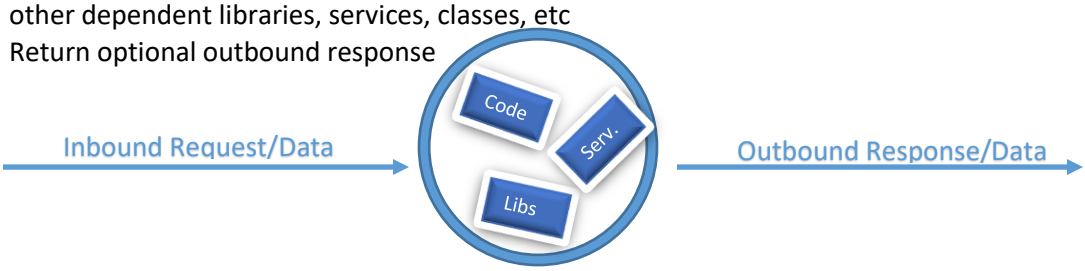
A^3 (Three Layer Abstraction) Programming Framework

Problem Statement – Code Inter-operability

Majority of the business code, irrespective of programming language, could grouped into three stages

- Interpret optional inbound request
- Process inbound request data through business logic. For processing/persistence, it refer to other dependent libraries, services, classes, etc
- Return optional outbound response



However, **porting code from one platform to another becomes a mammoth challenge as all three stages in some form or the other, are tightly coupled**.

**For example, business code using on-prem persistence storage, when ported to cloud and want to use cloud native persistence storage, it require changes in your business code.*

Solution - A^3 (Three Layer Abstraction) Programming Framework

With A^3 programming paradigm, all three stages (as defined below) are abstracted from business code, making code portability across platform a smooth experience.

- Layer 1 Abstraction – From inbound Request format to business code payload + business code output to outbound response format
- Layer 2 Abstraction – Dependent services and libraries referred by business code
- Layer 3 Abstraction – Instantiating dependent services and libraries referred by business code

Implementation - A^3 Programming Framework In Java

A^3 programming framework is available in java programming language supporting JRE version 1.5 or later

Implementation - A^3 Programming Framework For AWS Cloud

Using java based A^3 programming paradigm framework, we have built A^3 programming framework for AWS cloud with following implementation

- Layer 1 Abstraction
  - All supported AWS Lambda triggered event type such as API Gateway, SNS, SQS, DynamoDb Stream, etc to A^3 payload
  - A^3 response in json format
- Layer 2 Abstraction
  - Majority of commonly used AWS service i.e. AWS SNS, SQS, S3, DynamodDB, SES, Kinesis, Secret Manager and State Machine (step functions)
- Layer 3 Abstraction
  - Instantiating AWS services listed in "Layer 2 Abstraction" using AWS Resource ARN.
- CI/CD Pipeline
  - Pipeline to deploy dependent services/jar/libs as AWS Layer
  - Pipeline to deploy AWS Lambda
  - Reference Cloudformation template (Infra-as-code) for provisioning AWS services listed in Layer 2 abstraction

Benefits

- Developers need to focus only on developing business code, rest all plumbing from intercepting request, transformation, creating and injecting cloud services in business code, etc is taken care by framework, hence **increase developer productivity**
- Developer can start work on aws-cloud native development using AWS Lambda without having AWS cloud knowledge, hence **take care of challenge of training people on AWS**
- Business code remains cloud-agnostic and could be migrated across cloud providers such as AWS, Azure, etc with minimal or no changes, hence **provide clear exist strategy**
- Provide developers a reference cloud native development methodology, hence **bring development uniformity across teams**
- Flexibility **to write custom implementation across abstraction layer**. For example, writing layer 2 custom implementation for on-prem mongo db