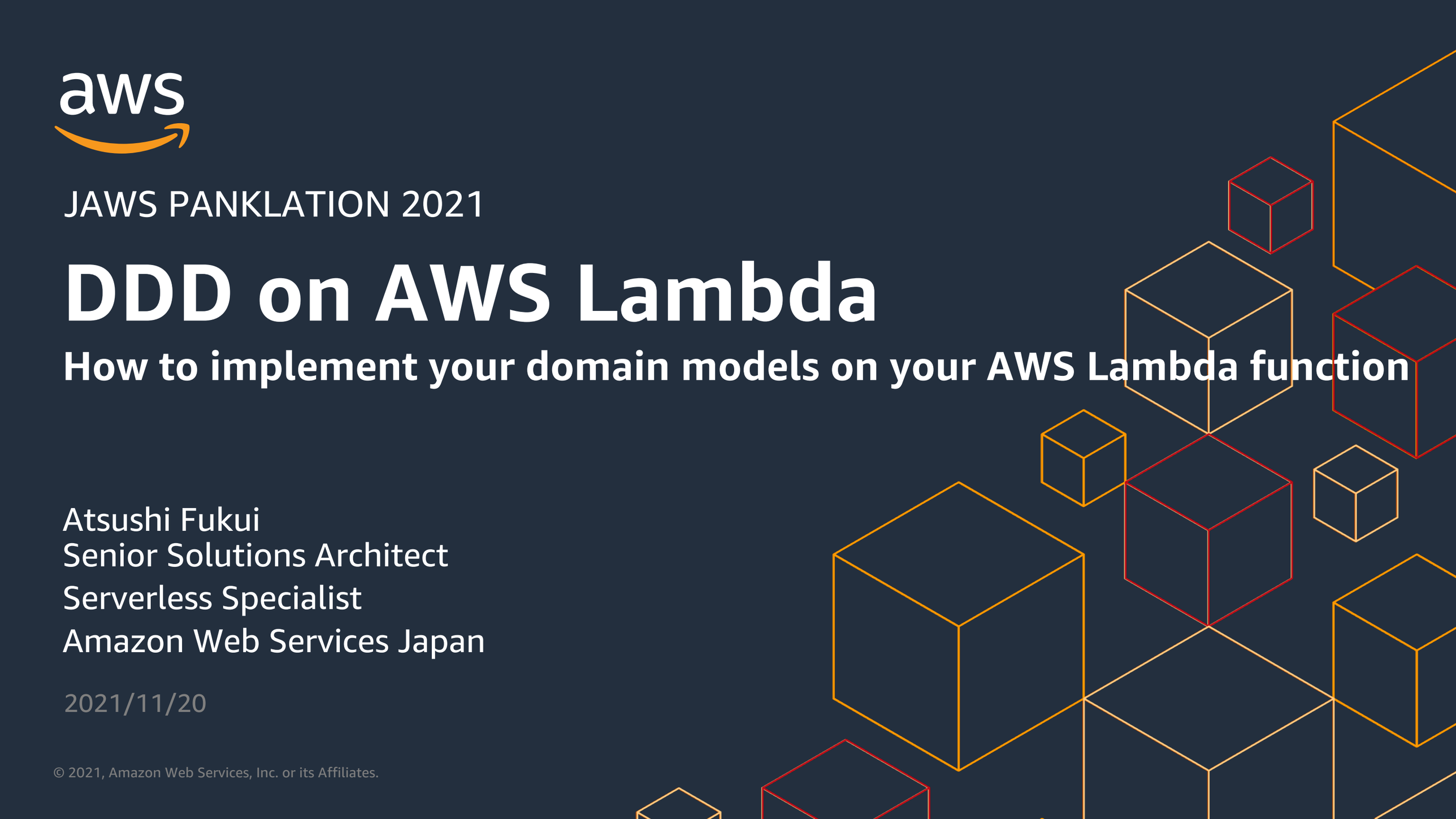
福井 厚

Atsushi Fukui

Amazon Web Services Japan

#jawsug #jawspankration2021 #jawspankration #aws



JAWS PANKLATION 2021

DDD on AWS Lambda

How to implement your domain models on your AWS Lambda function

Atsushi Fukui
Senior Solutions Architect
Serverless Specialist
Amazon Web Services Japan

2021/11/20

Who am I

- Name
  - Atsushi Fukui / twitter: afukui@
- Role and company
  - Senior Solutions Architect Serverless Specialist
  - Amazon Web Services Japan
- Interests
  - Software Architecture, Object Oriented Design and programming, Agile Development process
- My favorite AWS Services
  - AWS Lambda, AWS Step SQS and other all serverless services

aws

Agenda

- Domain Driven Design (DDD)
- Hexagonal Architecture
- Domain models for sample application
- How to implement unit testing code
- Demo
- Conclusion

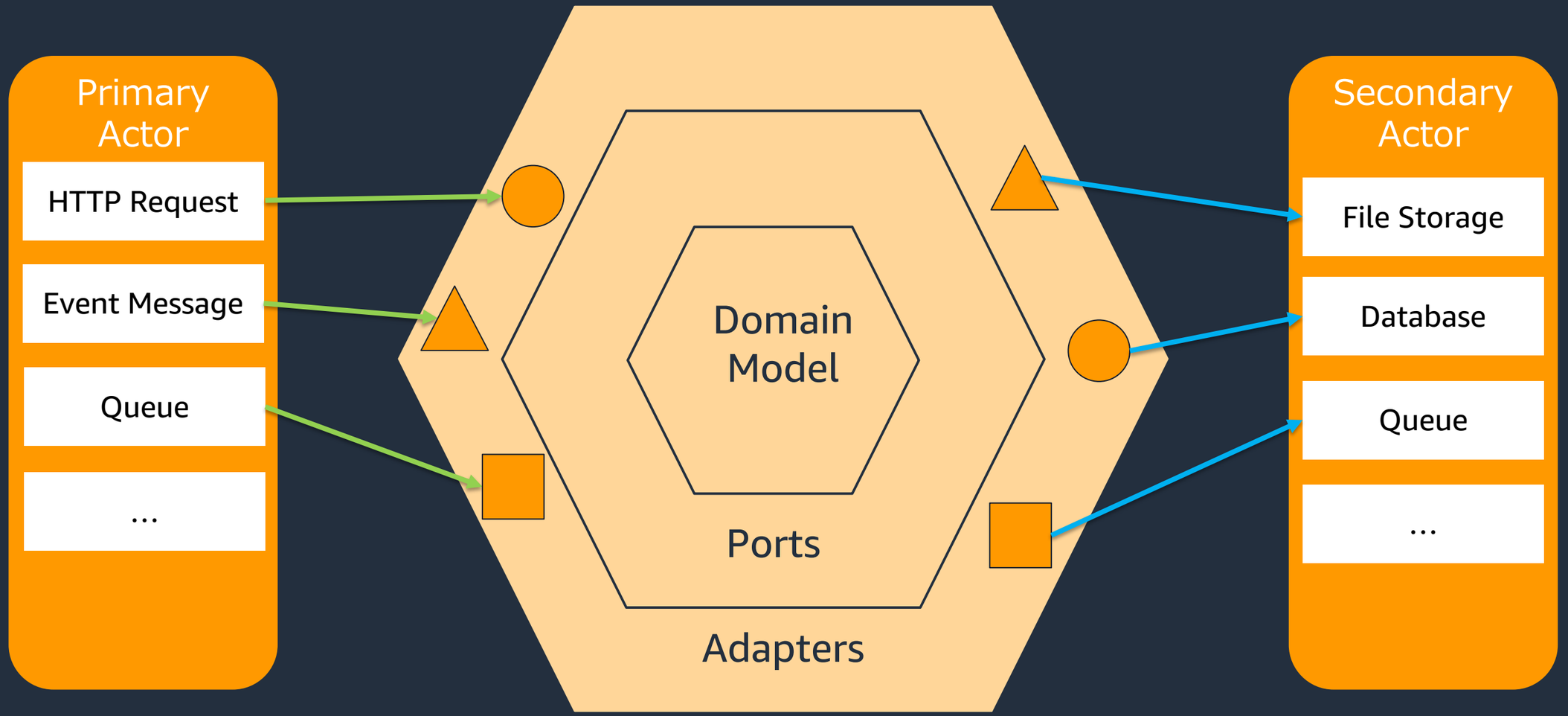
aws

Domain-driven design

- Provides a broad framework for making design decisions and developing a technical vocabulary for discussing domain design
- Ubiquitous language - modeling the language of the business
- Provides guidance about tactical design - model domains with entities, value objects, repositories and services, and strategic design…

aws

Hexagonal architecture



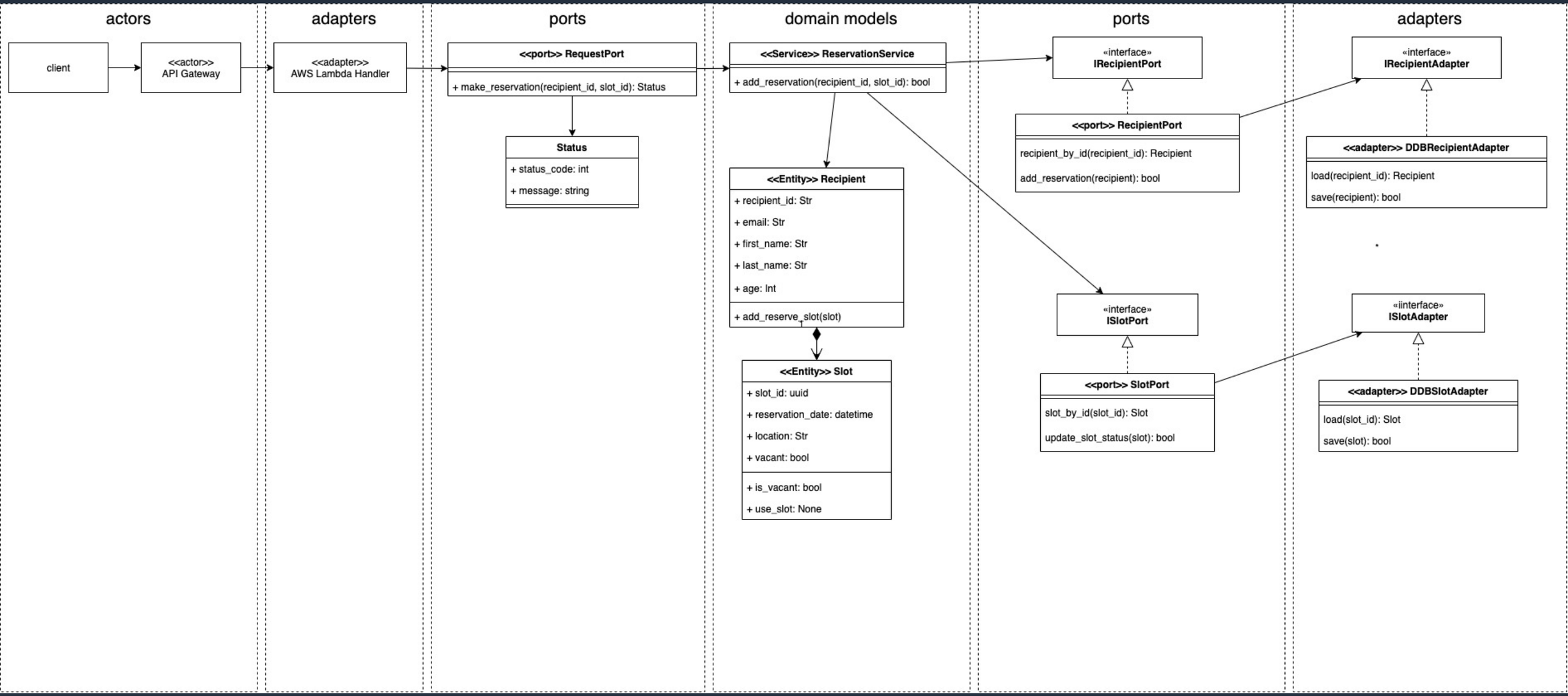
aws

Sample Senario

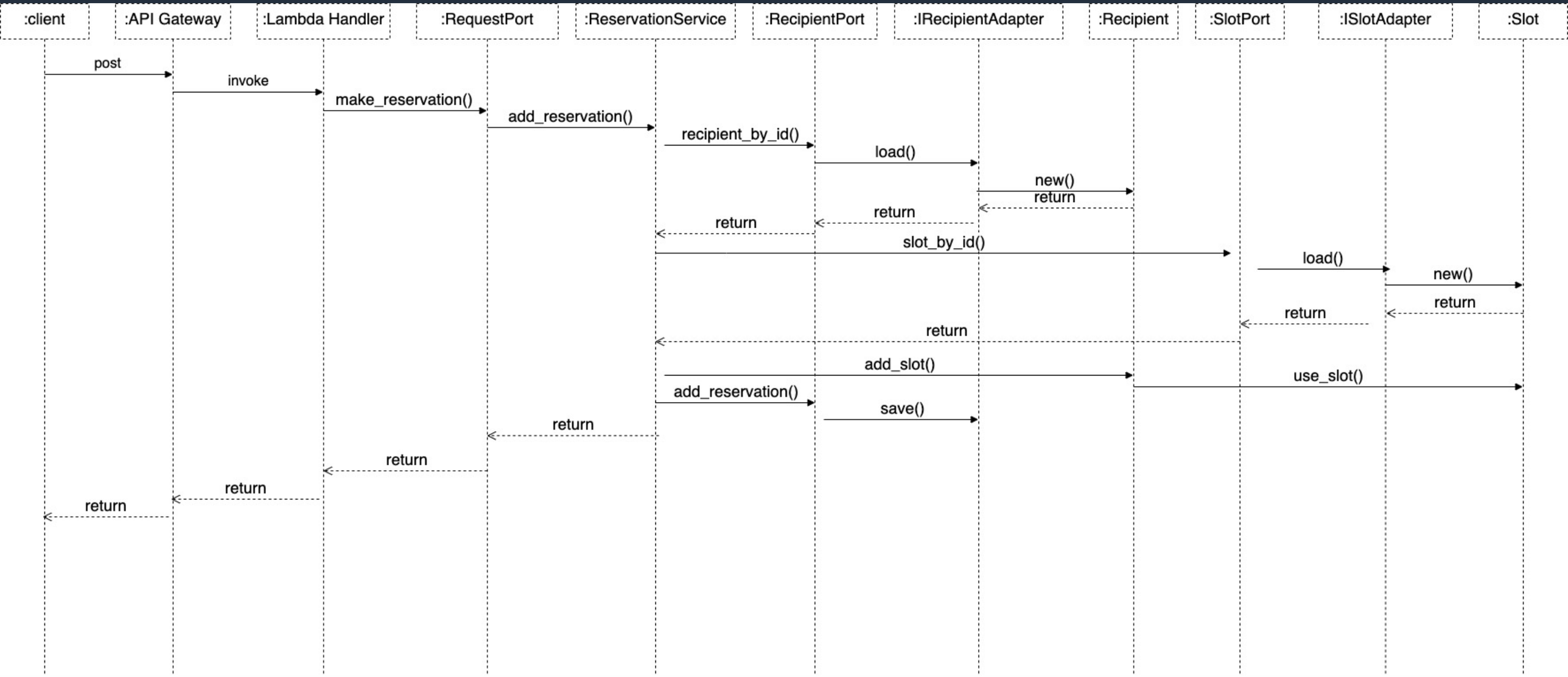
- Vaccination reservation system
  - Use cases:
    - A recipient can search for some vacant slots and register a reservation slot for vaccination reservation.
    - A recipient cannot register her reservation if there is no vacant slot.
    - A recipient cannot reserve her reservations more than two slots.
    - A recipient cannot reserve two reservations if there are same date time.

There are pure business logics you need to implement on your domain model.

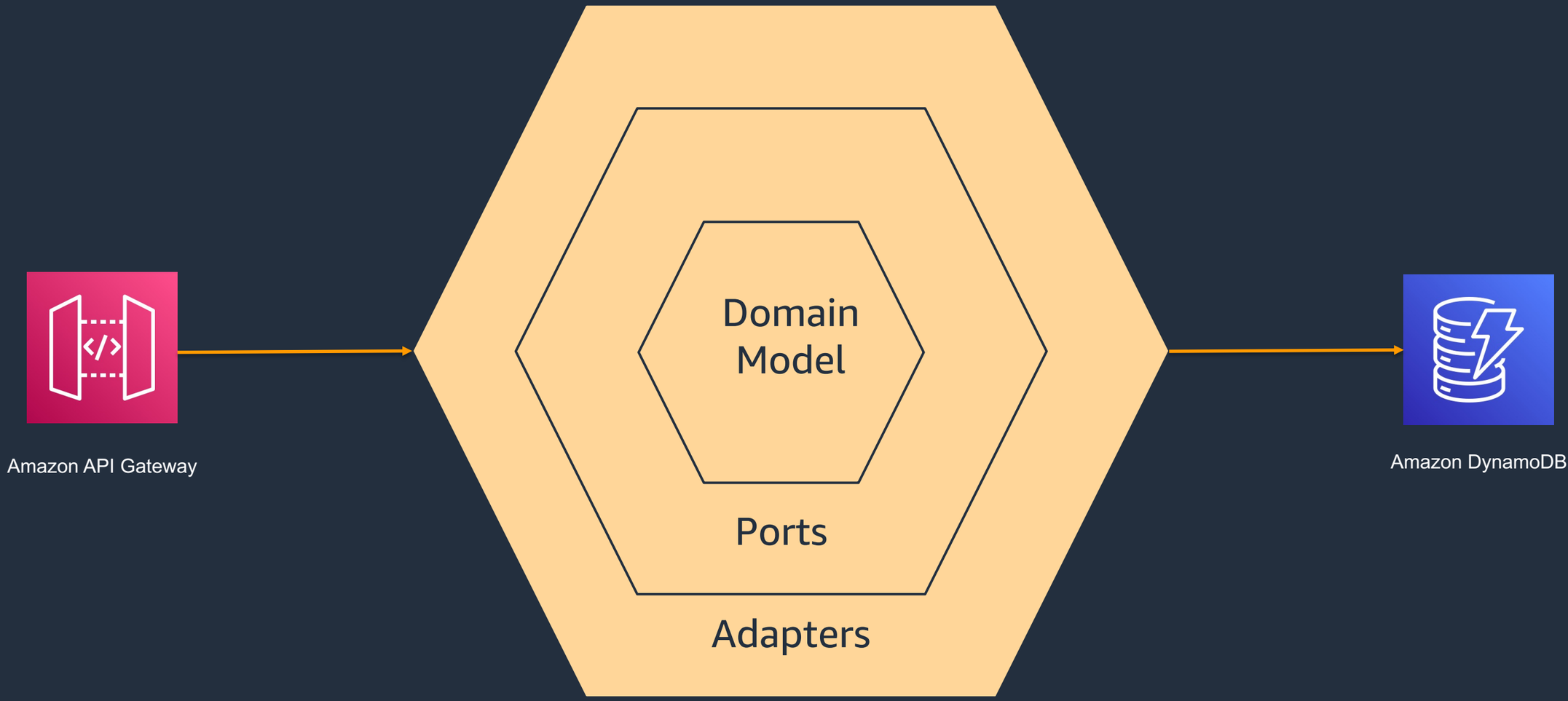
Class diagram: register vaccination reservation



Sequence diagram: register vaccination reservation



Hexagonal architecture with the Lambda function



Unit test for Domain Model

```
def test_cannot_append_slot_more_than_two(fixture_recipient, fixture_slot, fixture_slot_2, fixture_slot_3):
    slot = fixture_slot
    slot2 = fixture_slot_2
    slot3 = fixture_slot_3
    target = fixture_recipient
    target.add_reserve_slot(slot)
    target.add_reserve_slot(slot2)
    ret = target.add_reserve_slot(slot3)
    assert False == ret
    assert 2 == len(target.slots)

def test_cannot_append_same_date_slot(fixture_recipient, fixture_slot):
    slot = fixture_slot
    target = fixture_recipient
    target.add_reserve_slot(slot)
    ret = target.add_reserve_slot(slot)
    assert False == ret
    assert 1 == len(target.slots)
```


aws

Unit test for Ports and Adapters classes

```
class DummyRecipientAdapter(IRecipientAdapter):
    def load(self, recipient_id:str) -> Recipient:
        return Recipient(recipient_id, email, first_name, last_name, age)

    def save(self, recipient:Recipient) -> bool:
        return True
```

```
class DummyModule(Module):
    def configure(self, binder):
        binder.bind(RecipientPort, to=RecipientPort(DummyRecipientAdapter()))
```

```
@pytest.fixture()
def fixture_recipient_port():
    injector = Injector([DummyModule])
    recipient_port = injector.get(RecipientPort)
    return recipient_port
```


aws

Unit test for Ports and Adapters classes

```
def test_recipient_port_recipient_by_id(fixture_recipient_port):
    target = fixture_recipient_port
    recipient_id = "dummy_number"
    recipient = target.recipient_by_id(recipient_id)
    assert recipient != None
    assert recipient_id == recipient.recipient_id
    assert email == recipient.email
    assert first_name == recipient.first_name
    assert last_name == recipient.last_name
    assert age == recipient.age
```


aws

app.py calls concrate class instances

```
class RequestPortModule(Module):
    def configure(self, binder):
        binder.bind(ReservationService, to=ReservationService(
            RecipientPort(DDBRecipientAdapter()), SlotPort(DDBSlotAdapter())))
```

```
def lambda_handler(event, context):
body = json.loads(event['body'])
    recipient_id = body['recipient_id']
    slot_id = body['slot_id']

    injector = Injector([RequestPortModule])
    request_port = injector.get(RequestPort)
    status = request_port.make_reservation(recipient_id, slot_id)
```


aws

Demo

aws

Summary

- Loosely coupling and strong encapsulation are key concept.
- Hexagonal architecture is an architecture pattern used for encapslating domain logic, and decoupling it from other implementaion details, such as infrastructure or client requests.
- This approach can help create separetion of concerns and sepalate the domain logic from the infrastructure.
- Inversion of control (IoC) or injecting object instances is useful for unit testing with mock or fake objects.

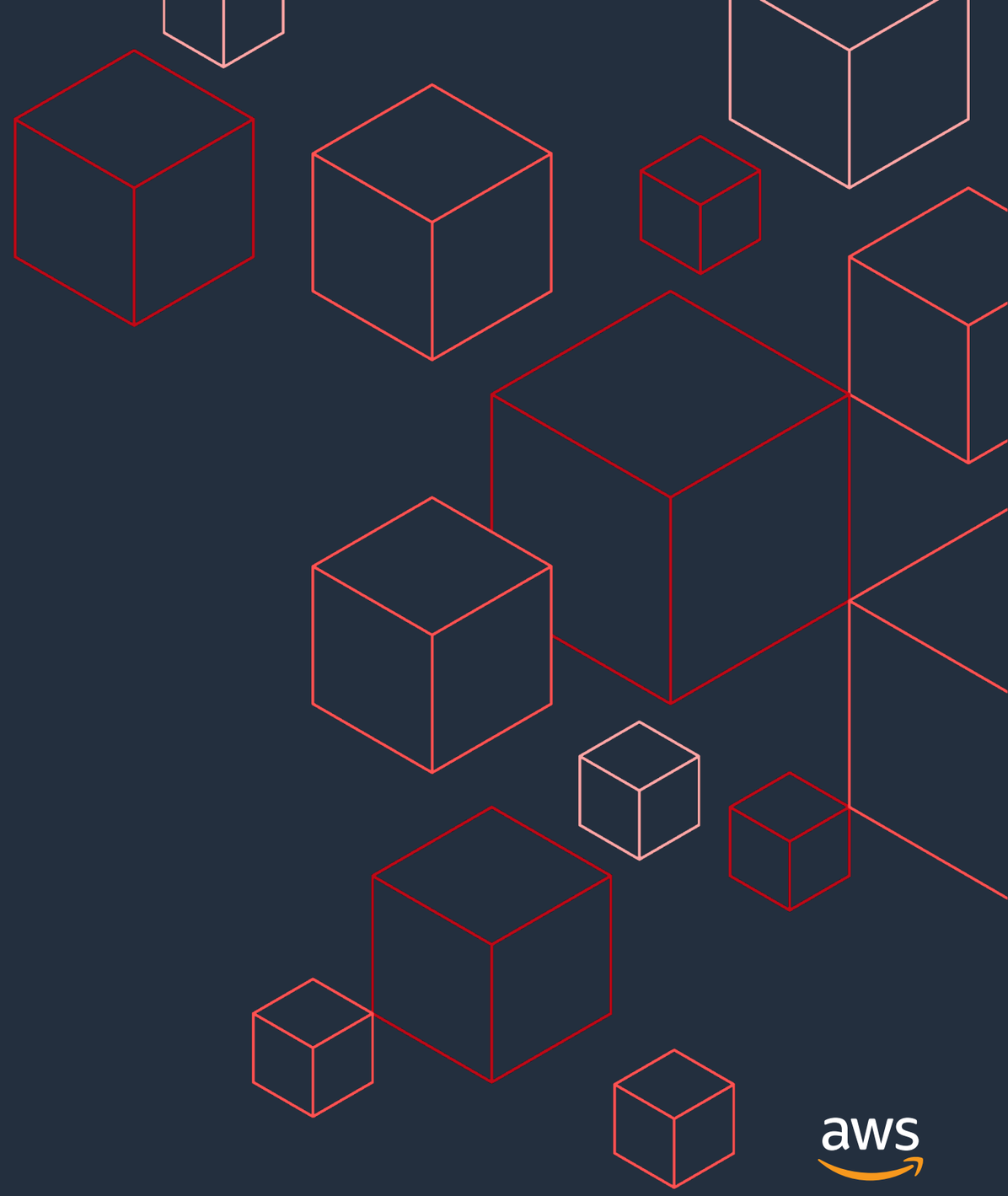
aws

Please see my sample project on GitHub

- https://github.com/afukui/jaws-pankration-ddd-lambda

aws

Thank you!



aws