AWS re:Invent

NOV. 28 – DEC. 2, 2022 | LAS VEGAS, NV

COM205

Let's write a microservice using AWS Lambda (or maybe not)

Joanne Skiles (she/her)

Engineering Leader | AWS Community Builder

Agenda

- The problem (i.e., The introduction)
- Designing a microservice using AWS Lambda
- Implementing and deploying the solution
- Pros and cons
- Takeaways

The problem

(i.e., The introduction)

Background

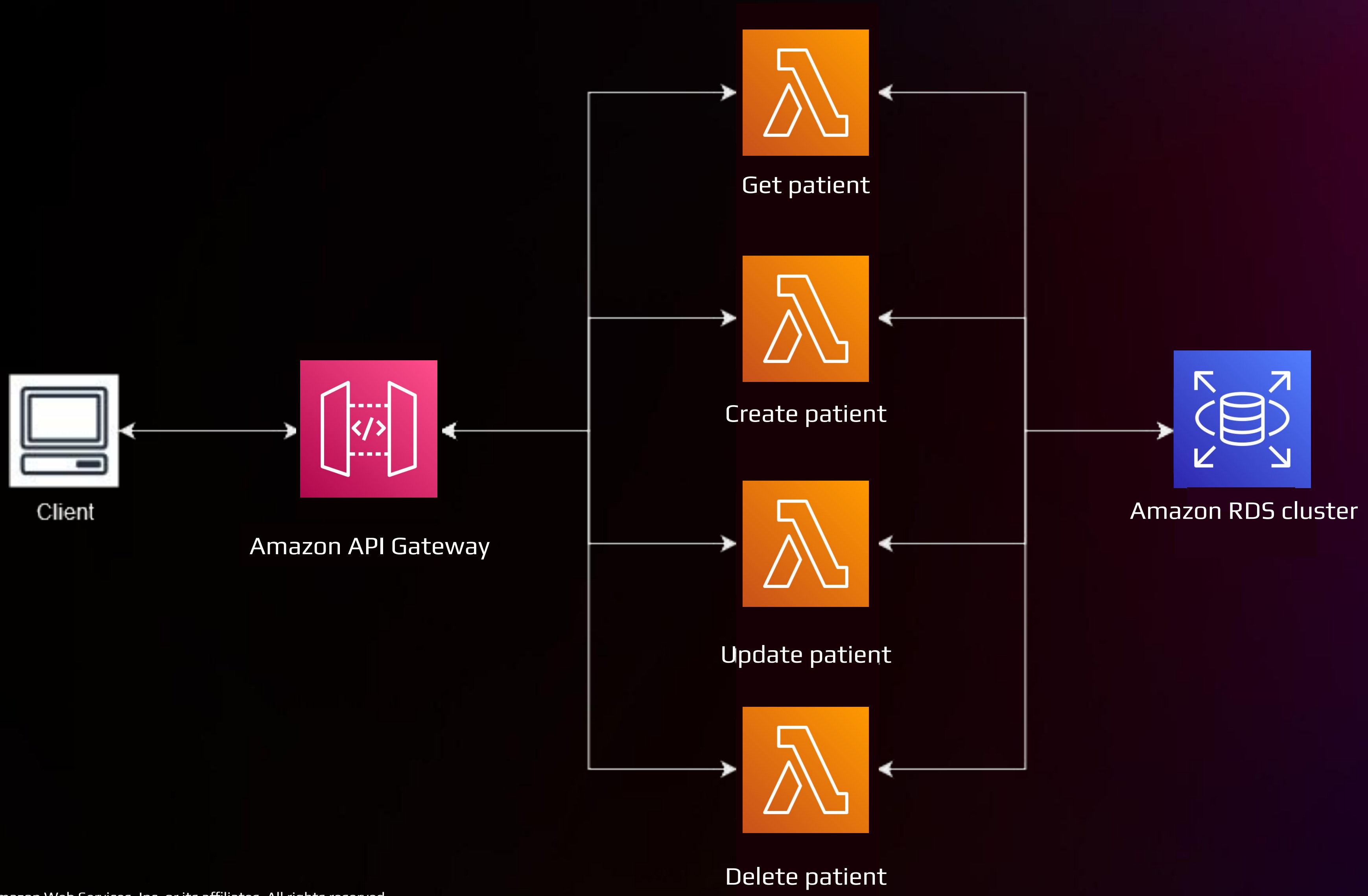
- You work at generic healthtech startup company
- You’ve been tasked with building a microservice to handle all patient information
- Instead of a traditional microservice, could you build it using AWS Lambda?

Quick review

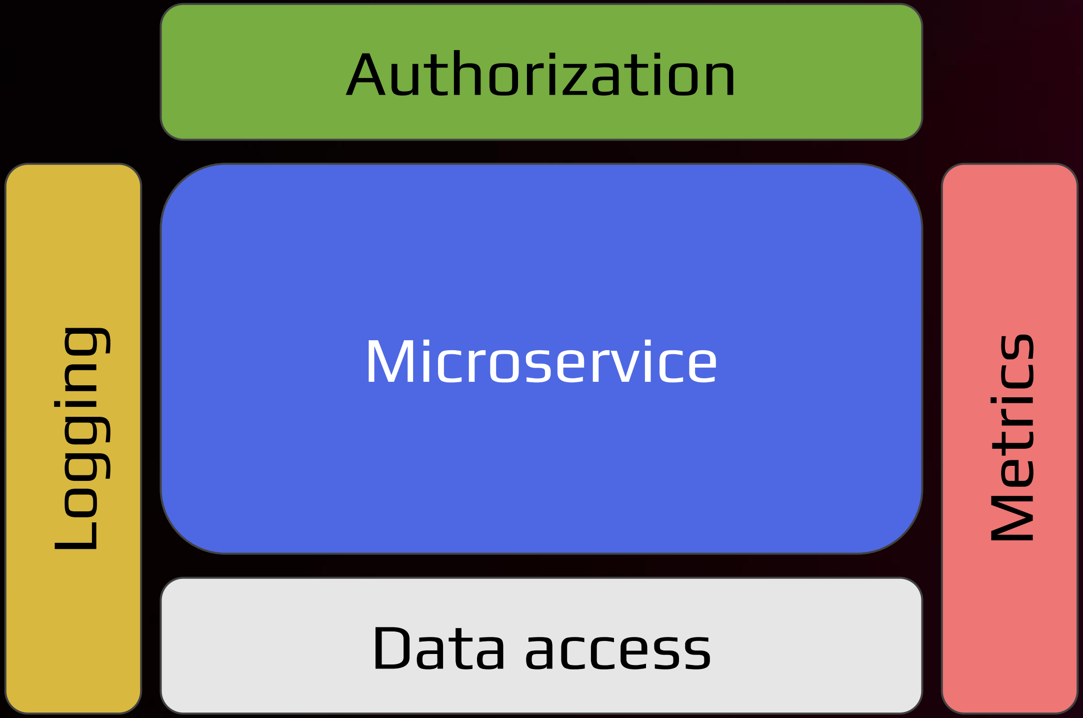
- Event-based architecture
  - Uses events to trigger and communicate between decoupled services
    - Event is a change in state or an update
    - Has three key components
      - Event producers
      - Event routers
      - Event consumers
- AWS Lambda
  - Integrates with other AWS services to invoke functions based on events that you specify
  - Compute service that lets you run code (function) only when needed and scales automatically

Designing a microservice using AWS Lambda

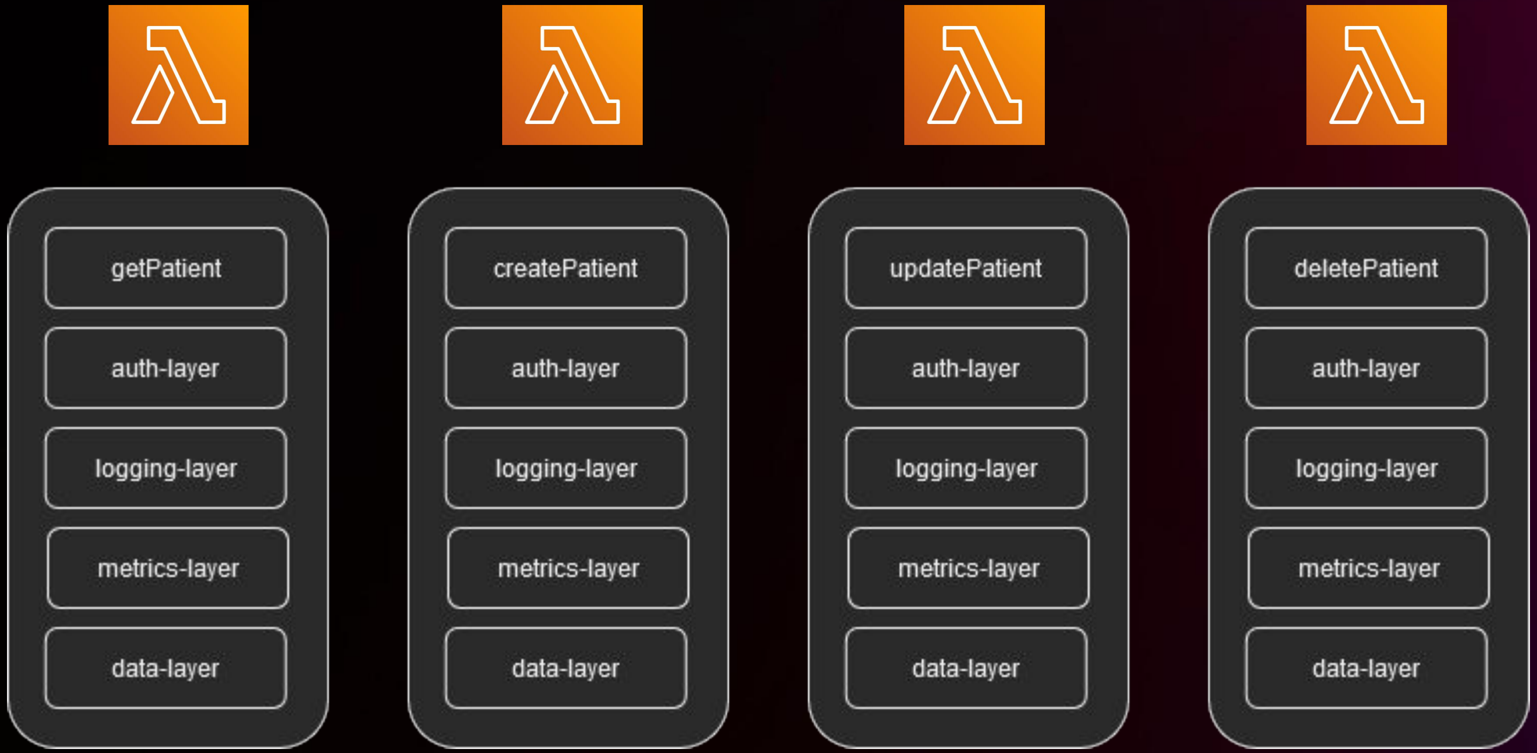
Designing a microservice using AWS Lambda



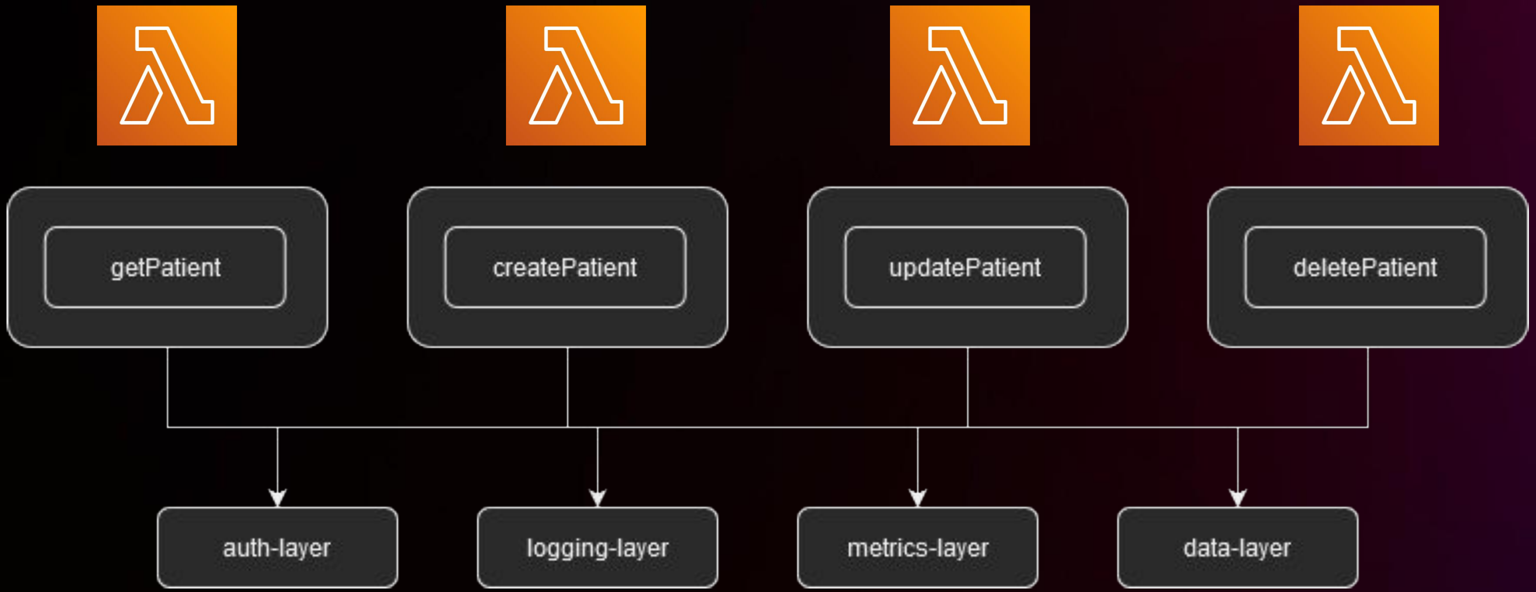
Common components



AWS layers


getPatient
auth-layer
logging-layer
metrics-layer
data-layer
createPatient
auth-layer
logging-layer
metrics-layer
data-layer
updatePatient
auth-layer
logging-layer
metrics-layer
data-layer
deletePatient
auth-layer
logging-layer
metrics-layer
data-layer

AWS layers


getPatient
createPatient
updatePatient
deletePatient
auth-layer
logging-layer
metrics-layer
data-layer

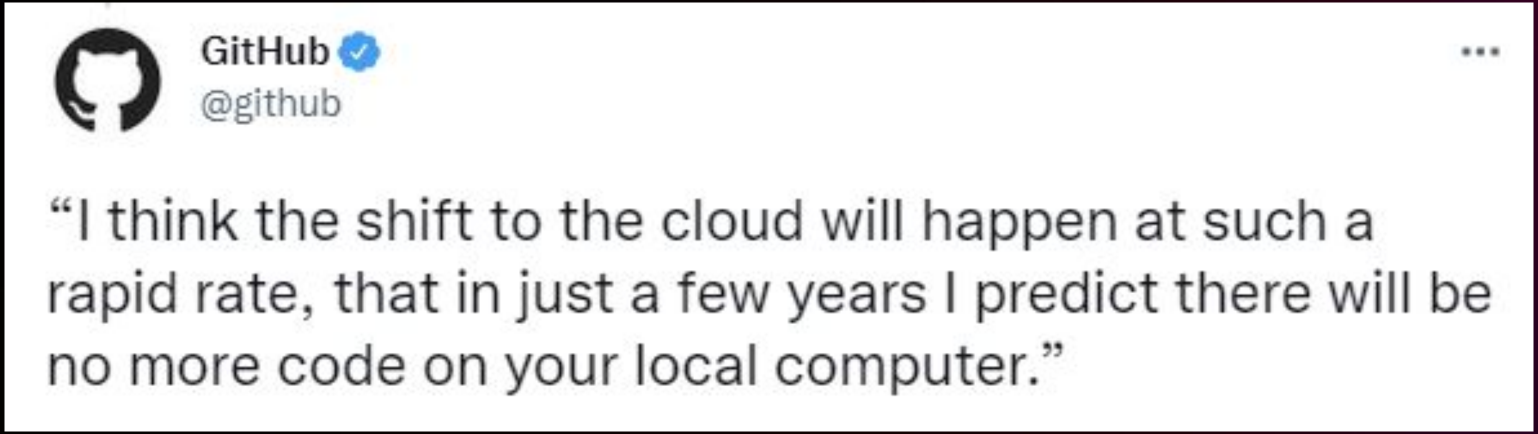
Implementing and deploying the solution

Pros and cons

Pros

- Low cost
- Improves scalability
- Highly available

Cons


GitHub
@github

"I think the shift to the cloud will happen at such a rapid rate, that in just a few years I predict there will be no more code on your local computer."

GitHub, Twitter post, June 15, 2022 https://bit.ly/3DFSFhV

Cons

- No local development
- Your framework and code **may** need to change
- Automatic scaling can lead to DoS against yourself
- Upgrades can be challenging
- Vendor lock-in
- Lambda quotas

GitHub, Twitter post, June 15, 2022 https://bit.ly/3DFSFhV

Takeaways

Not a general purpose solution

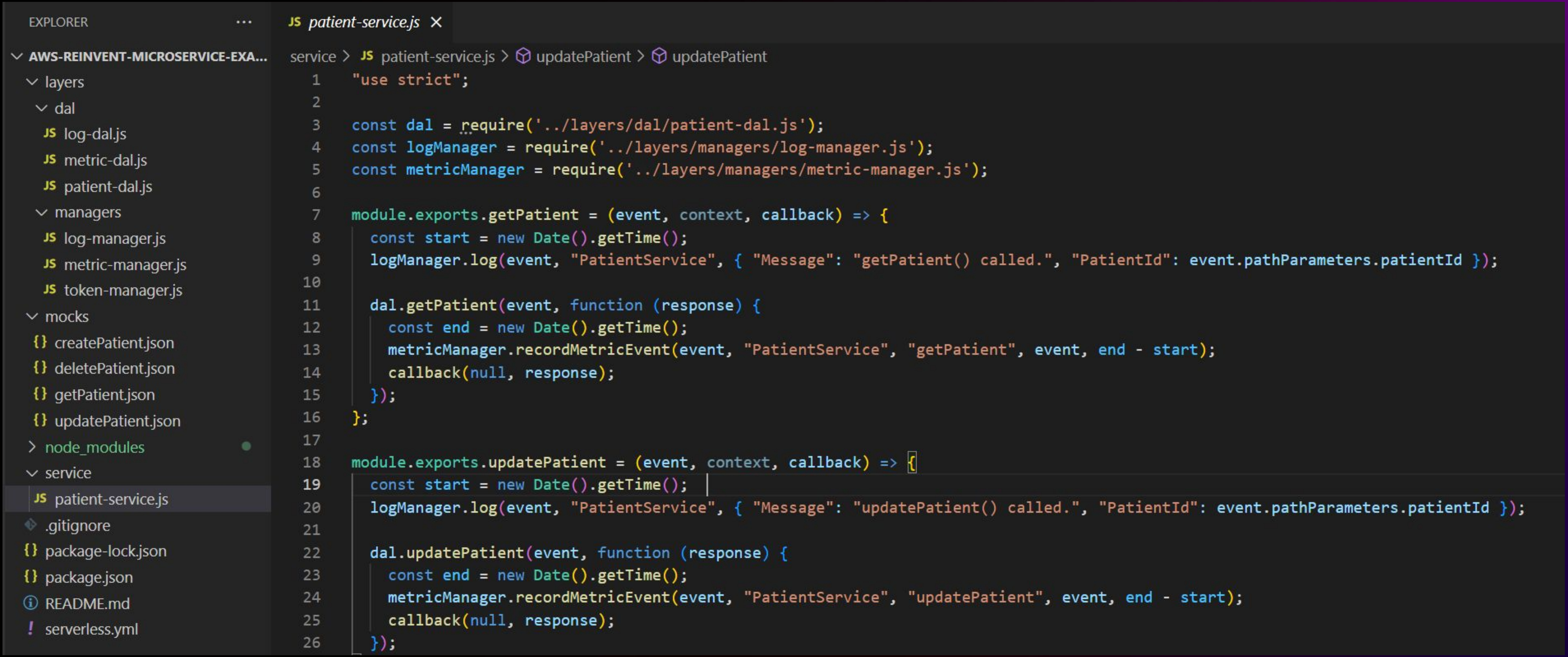
- Understand
  - The problem you are trying to solve
  - The knowledge of your team
  - What you value (flexibility, control, options, etc.)
- Lambda is not appropriate for use cases such as
  - High availability services
  - UI apps that are pure static content
  - WebSocket apps
  - Big data apps
  - Heavy mathematical computation
  - Systems that need to be always on and stateful

Thank you!

Joanne Skiles
@drjoanneskiles
chaotictoejam
Dr. Joanne Skiles

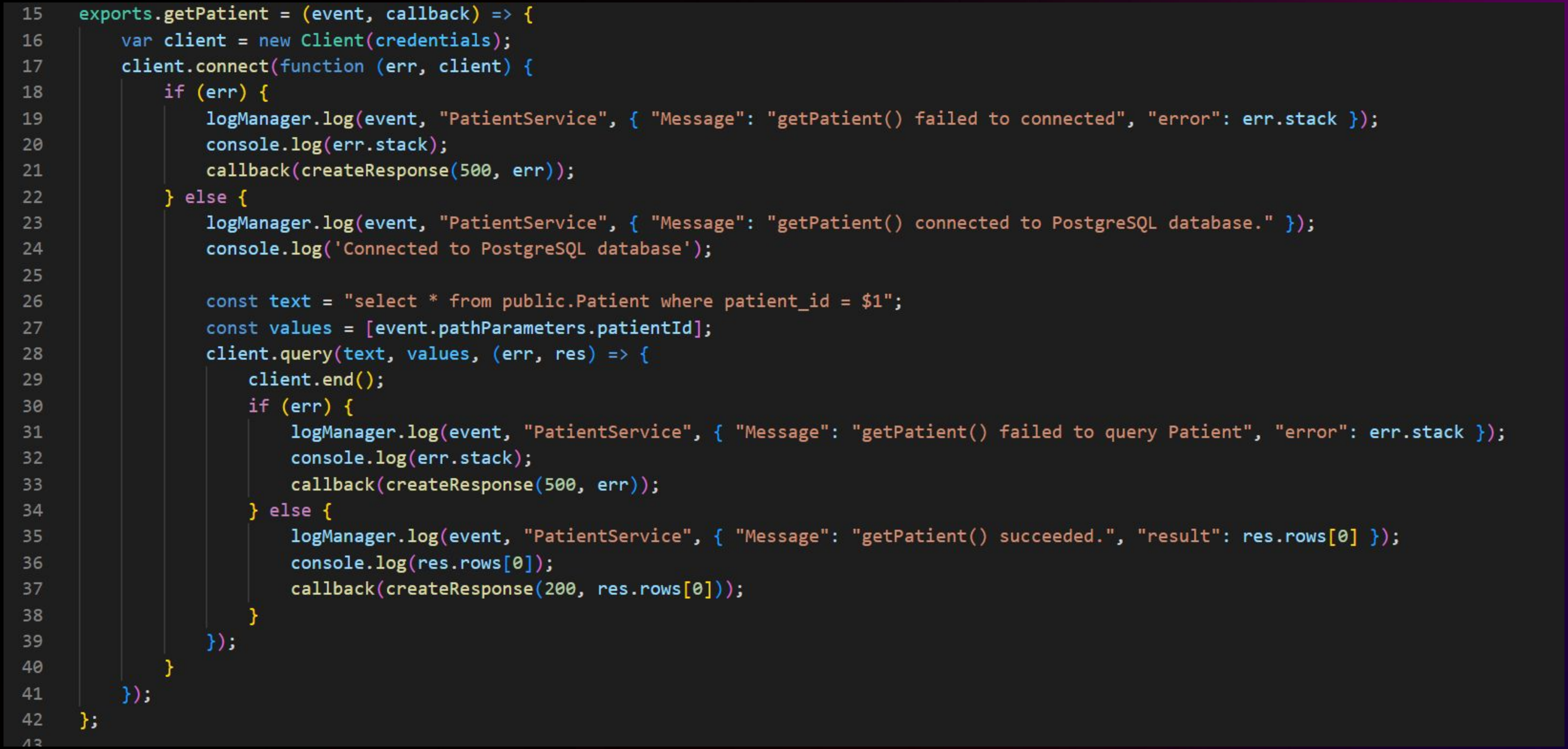
Please complete
the session survey

Handler (Get)



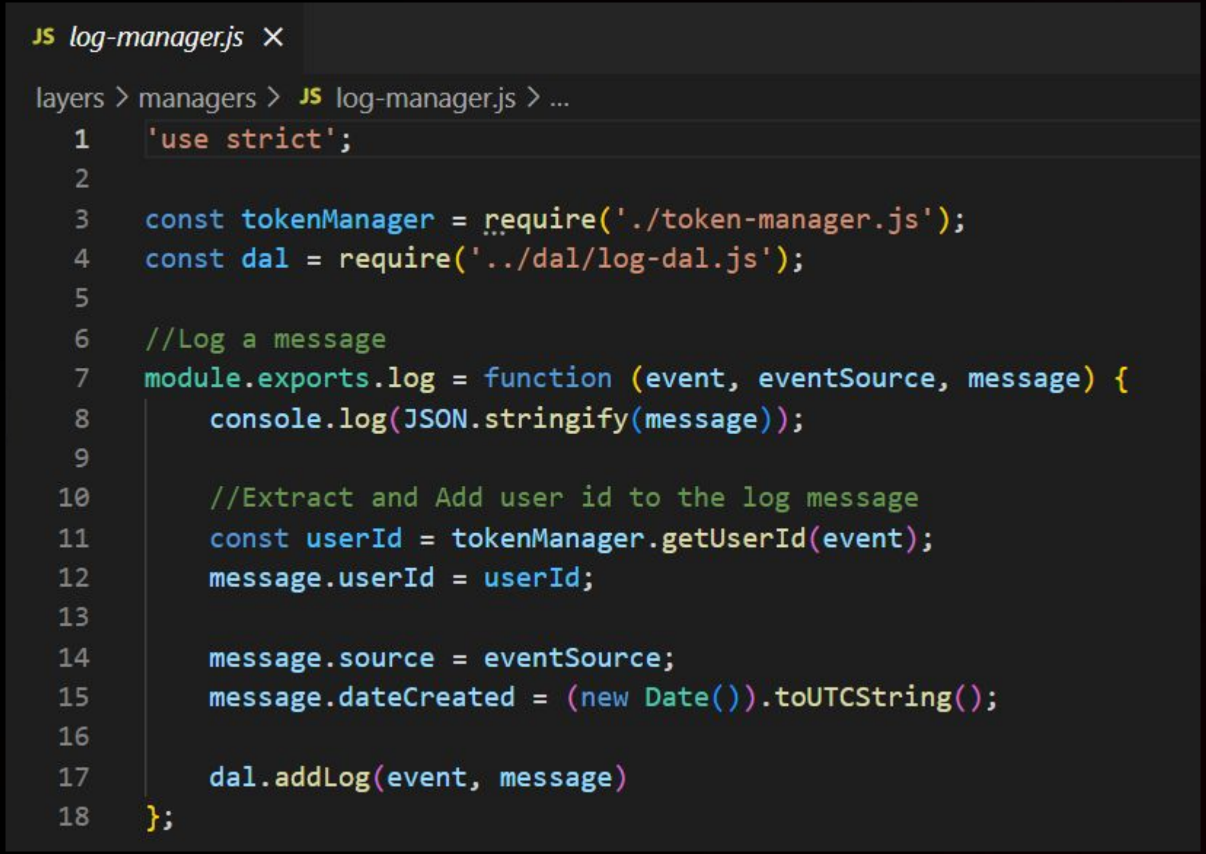
```javascript
"use strict";

const dal = require('../layers/dal/patient-dal.js');
const logManager = require('../layers/managers/log-manager.js');
const metricManager = require('../layers/managers/metric-manager.js');

module.exports.getPatient = (event, context, callback) => {
  const start = new Date().getTime();
  logManager.log(event, "PatientService", { "Message": "getPatient() called.", "PatientId": event.pathParameters.patientId });

  dal.getPatient(event, function (response) {
    const end = new Date().getTime();
    metricManager.recordMetricEvent(event, "PatientService", "getPatient", event, end - start);
    callback(null, response);
  });
};

module.exports.updatePatient = (event, context, callback) => {
  const start = new Date().getTime();
  logManager.log(event, "PatientService", { "Message": "updatePatient() called.", "PatientId": event.pathParameters.patientId });

  dal.updatePatient(event, function (response) {
    const end = new Date().getTime();
    metricManager.recordMetricEvent(event, "PatientService", "updatePatient", event, end - start);
    callback(null, response);
  });
```

Patient DAL (Get)



```
exports.getPatient = (event, callback) => {
    var client = new Client(credentials);
    client.connect(function (err, client) {
        if (err) {
            logManager.log(event, "PatientService", { "Message": "getPatient() failed to connected", "error": err.stack });
            console.log(err.stack);
            callback(createResponse(500, err));
        } else {
            logManager.log(event, "PatientService", { "Message": "getPatient() connected to PostgreSQL database." });
            console.log('Connected to PostgreSQL database');

            const text = "select * from public.Patient where patient_id = $1";
            const values = [event.pathParameters.patientId];
            client.query(text, values, (err, res) => {
                client.end();
                if (err) {
                    logManager.log(event, "PatientService", { "Message": "getPatient() failed to query Patient", "error": err.stack });
                    console.log(err.stack);
                    callback(createResponse(500, err));
                } else {
                    logManager.log(event, "PatientService", { "Message": "getPatient() succeeded.", "result": res.rows[0] });
                    console.log(res.rows[0]);
                    callback(createResponse(200, res.rows[0]));
                }
            });
        }
    });
};
```

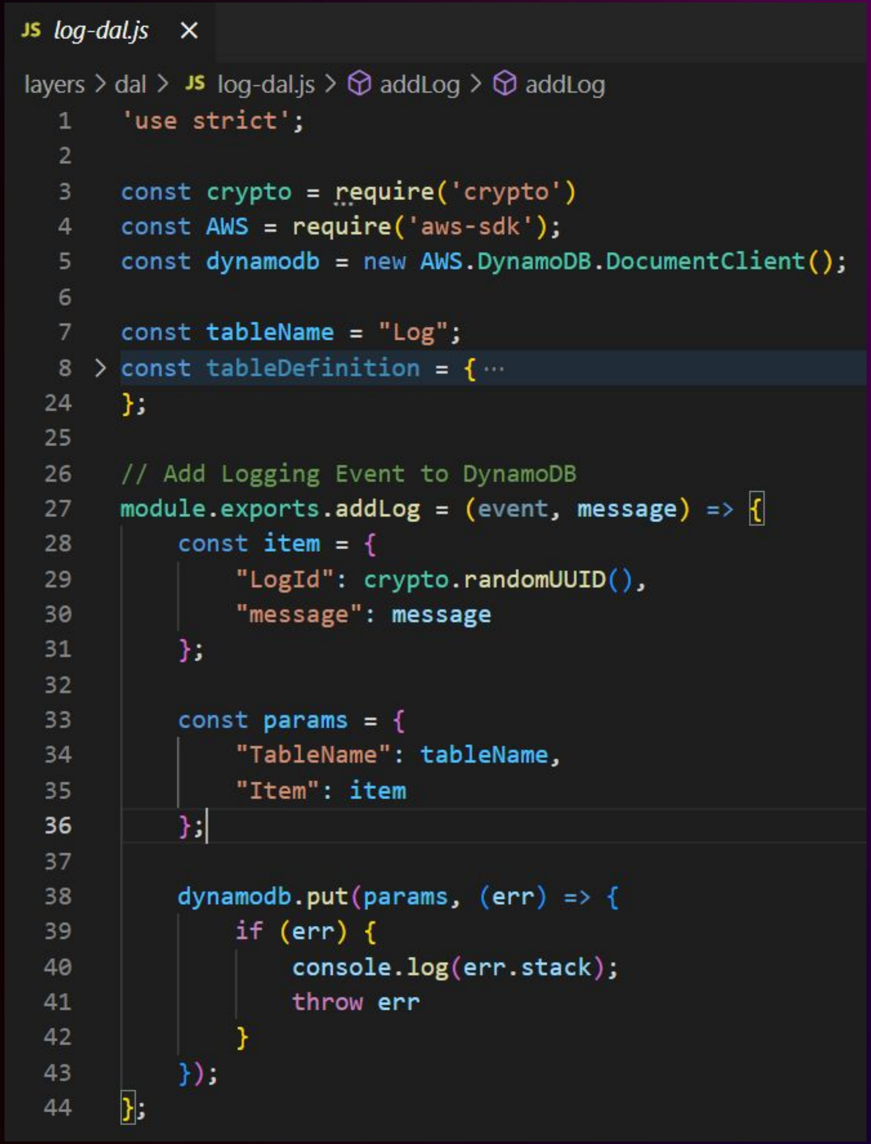
Log Manager & DAL



log-manager.js

layers > managers > log-manager.js > ...

```javascript
'use strict';

const tokenManager = require('./token-manager.js');
const dal = require('../dal/log-dal.js');

//Log a message
module.exports.log = function (event, eventSource, message) {
    console.log(JSON.stringify(message));

    //Extract and Add user id to the log message
    const userId = tokenManager.getUserId(event);
    message.userId = userId;

    message.source = eventSource;
    message.dateCreated = (new Date()).toUTCString();

    dal.addLog(event, message)
};
```

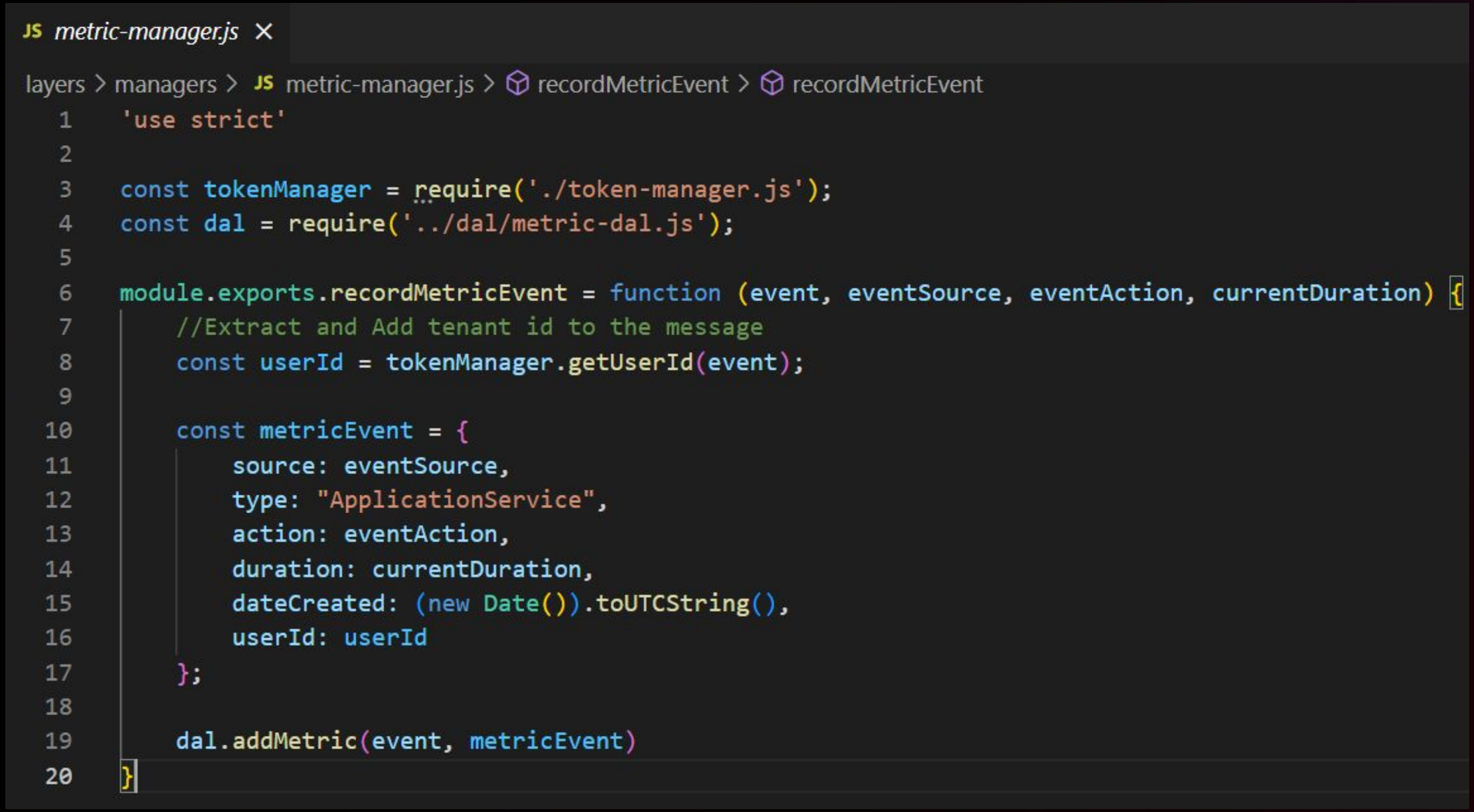


log-dal.js

layers > dal > log-dal.js > addLog > addLog

```javascript
'use strict';

const crypto = require('crypto')
const AWS = require('aws-sdk');
const dynamodb = new AWS.DynamoDB.DocumentClient();

const tableName = "Log";
const tableDefinition = {…
};

// Add Logging Event to DynamoDB
module.exports.addLog = (event, message) => {
    const item = {
        "LogId": crypto.randomUUID(),
        "message": message
    };

    const params = {
        "TableName": tableName,
        "Item": item
    };

    dynamodb.put(params, (err) => {
        if (err) {
            console.log(err.stack);
            throw err
        }
    });
};
```

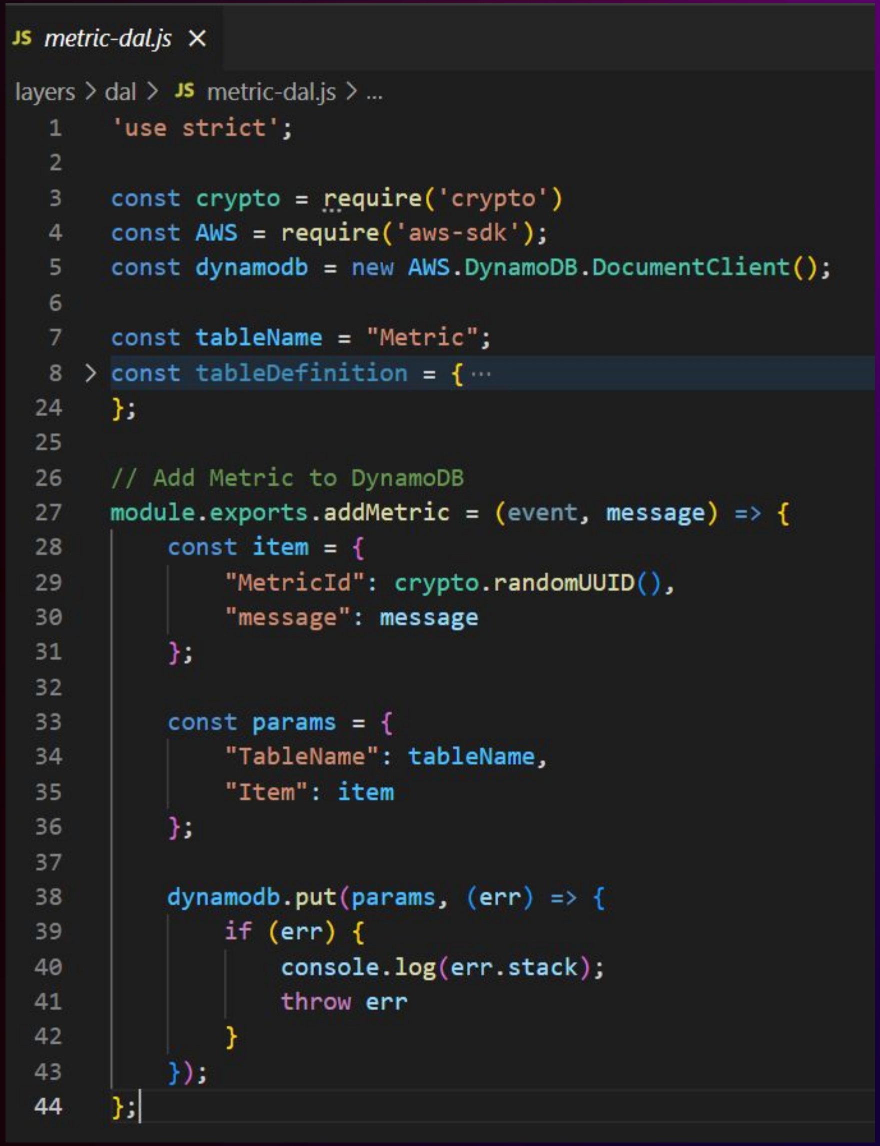
Metric Manager & DAL



metric-manager.js

layers > managers > metric-manager.js > recordMetricEvent > recordMetricEvent

```javascript
'use strict'

const tokenManager = require('./token-manager.js');
const dal = require('../dal/metric-dal.js');

module.exports.recordMetricEvent = function (event, eventSource, eventAction, currentDuration) {
    //Extract and Add tenant id to the message
    const userId = tokenManager.getUserId(event);

    const metricEvent = {
        source: eventSource,
        type: "ApplicationService",
        action: eventAction,
        duration: currentDuration,
        dateCreated: (new Date()).toUTCString(),
        userId: userId
    };

    dal.addMetric(event, metricEvent)
}
```



metric-dal.js

layers > dal > metric-dal.js > ...

```javascript
'use strict';

const crypto = require('crypto')
const AWS = require('aws-sdk');
const dynamodb = new AWS.DynamoDB.DocumentClient();

const tableName = "Metric";
const tableDefinition = {…
};

// Add Metric to DynamoDB
module.exports.addMetric = (event, message) => {
    const item = {
        "MetricId": crypto.randomUUID(),
        "message": message
    };

    const params = {
        "TableName": tableName,
        "Item": item
    };

    dynamodb.put(params, (err) => {
        if (err) {
            console.log(err.stack);
            throw err
        }
    });
};
```