Project Insight

MATTHEW CLEMENCE, JULIO BONILLA, RYAN DRAKE, DYLAN VAUGHN, LOGAN REED

Ponga

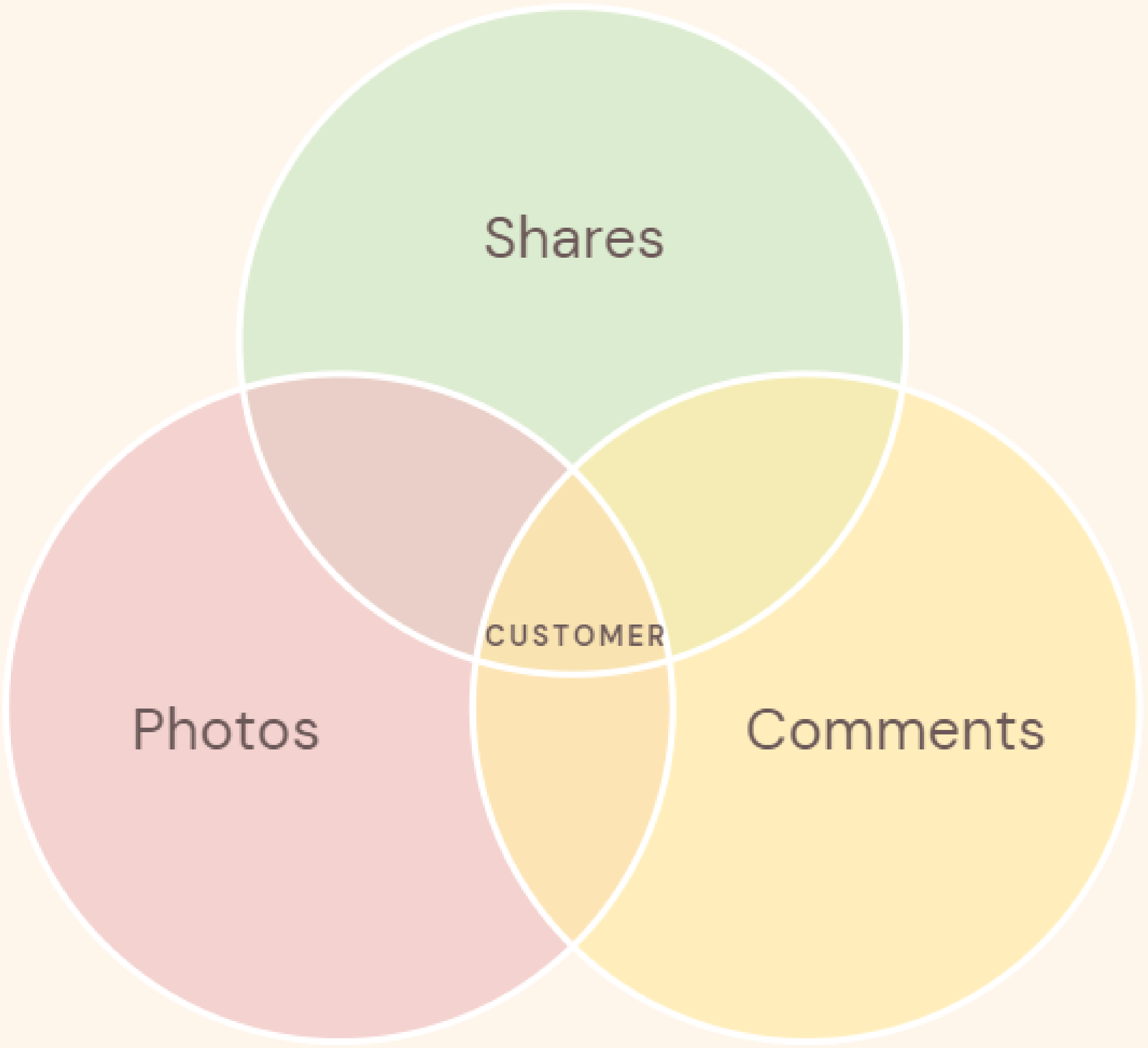
- What is Ponga?
- The Problem
- Project Insights First Stage
- Value Propositions

Value Propositions

- What are they?
- Reese's

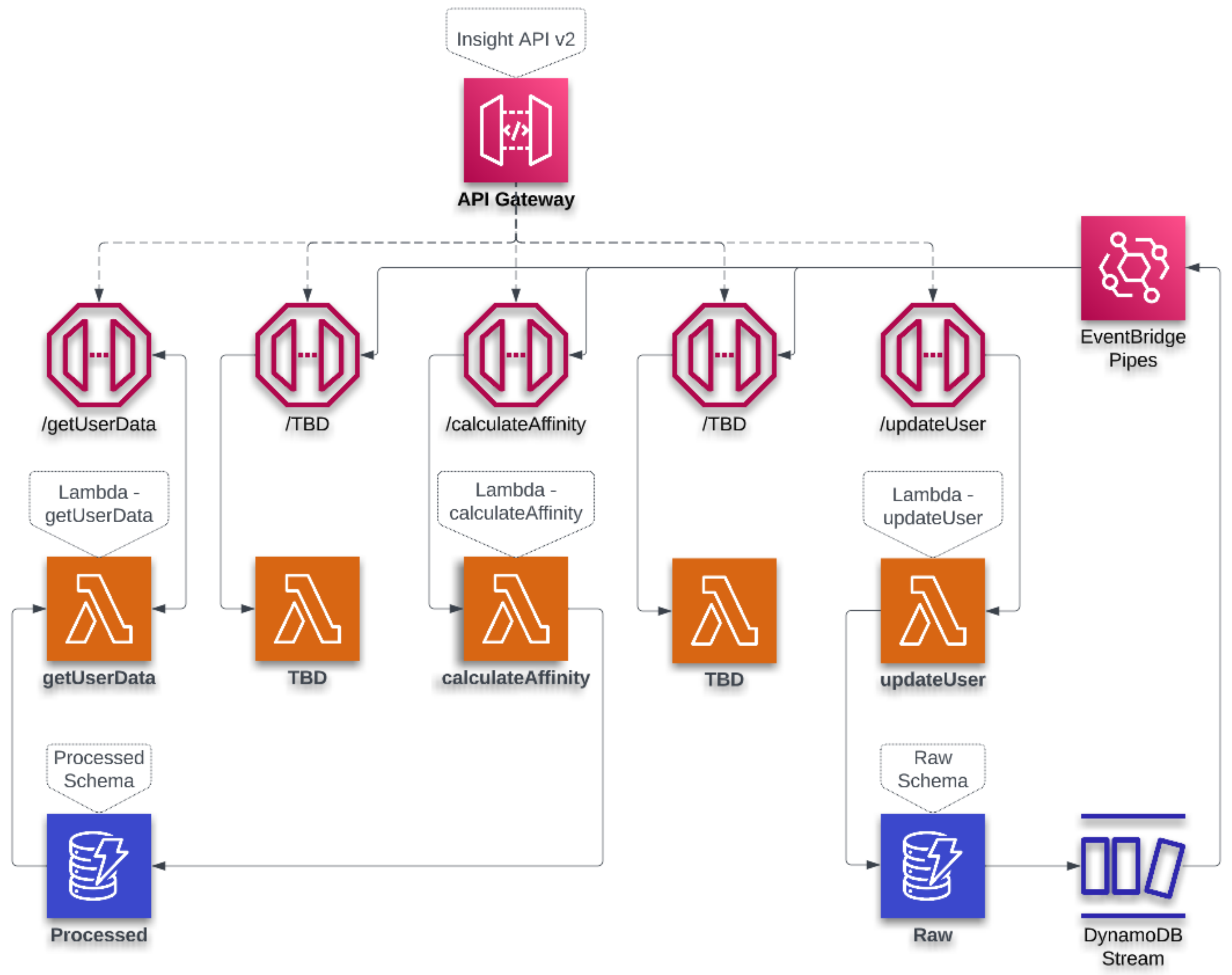
First Stage

- Collect Data
- Analyze and return
- Amazon Web Services:
  - API Gateway
  - EventBridge - Pipes
  - Lambda - Functions
  - DynamoDB



Project Prospects

- Define a healthy user
- Create user states
- Ponga takes action based on user's state
- Goal is self-run program


Insight API v2
API Gateway
EventBridge Pipes
/getUserData
/TBD
/calculateAffinity
/TBD
/updateUser
Lambda - getUserData
Lambda - calculateAffinity
Lambda - updateUser
getUserData
TBD
calculateAffinity
TBD
updateUser
Processed Schema
Raw Schema
Processed
Raw
DynamoDB Stream

Source

Receive events from a variety of sources, including DynamoDB, Kinesis, and SQS.

Filtering (optional)

Define an event pattern to filter the events that are sent through the pipe.

Enrichment (optional)

Transform your event or pull additional data into it using Lambda, Step Functions, or an API.

Target

Send your event to an Amazon service, an event bus, or an API destination.

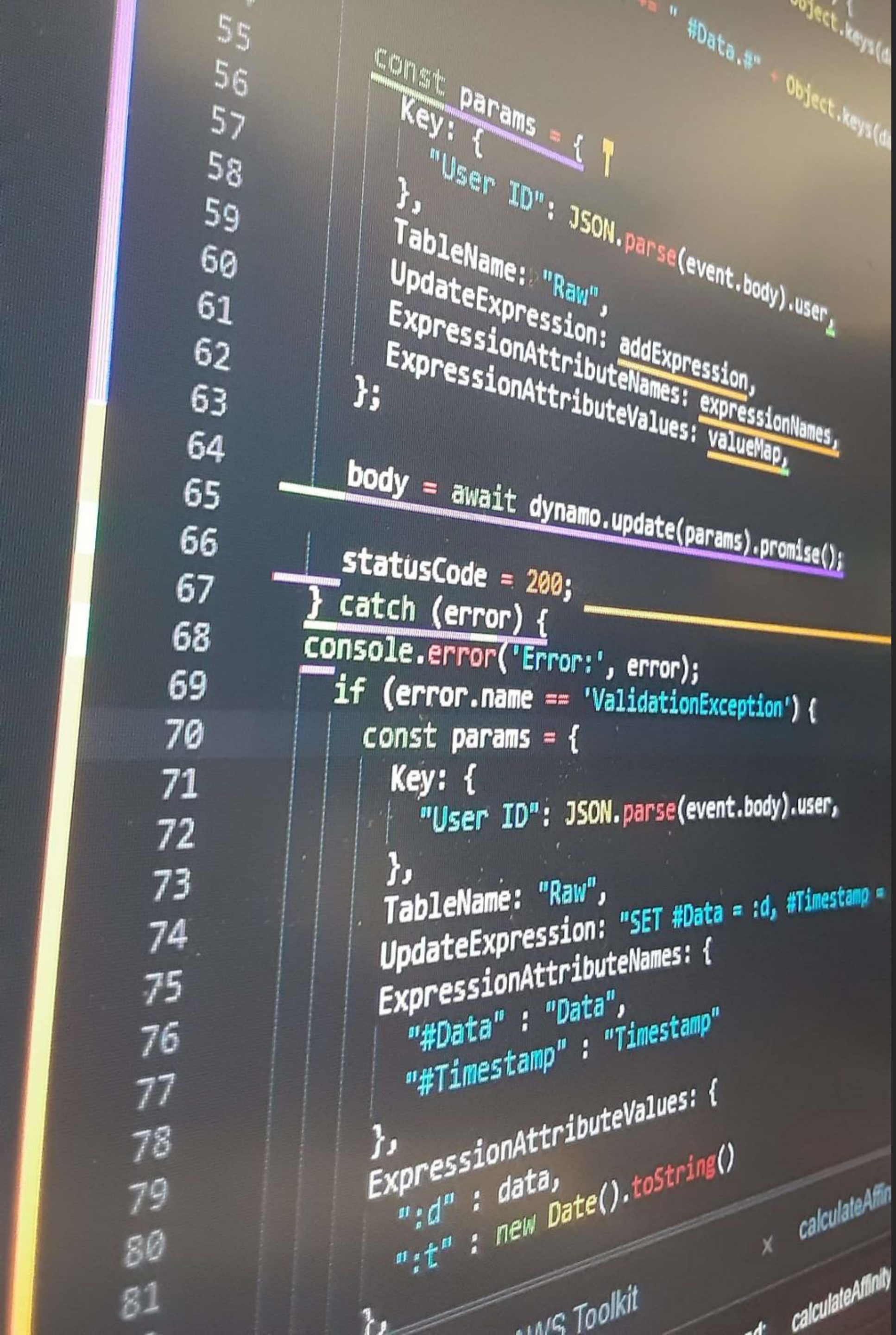
EventBridge Pipes

Captures, filters and delivers events.

```
                                            ws-sdk');
  dynamo = new AWS.DynamoDB.DocumentClient();
exports.handler = async (event, context) => {
    console.log(JSON.stringify(event, 0, null));

    const angles = [];
    var vectorSum = 0;
    var Data = JSON.parse(event.body).dynamodb.NewImage

    var numVar = Object.keys(Data).length;

    function calculateAngles() {
        let iter = 0;
        for (let i = 0; i < numVar; i++) {
            for (let j = i+1; j < numVar; j++) {

                let relationshipName = Object.keys(Data)[
                let angVal = Math.atan(Object.values(Data)[
                let dominantProp = "";

                /*/
                Determine the dominant proposition
                /*/
                if (angVal < 45) {
                    dominantProp = Object.keys(Data)[j].
                }
                else if (angVal > 45) {
                    dominantProp = Object.keys(Data)[i].
                }
                else {
                    dominantProp = "Equivalent";
                }
                    relationshipName,
```

Lambda - calculateAffinity

- Receives an event from an EventBridge pipe containing user data
- Calculates the magnitude and N angles determined by the equation N*(N-1)/2
- These angles express the relationships between each value proposition
- Stored in the processed table with an attribute called Dominant to improve readability



```
        " #Data.#" + Object.keys(d
    const params = {
      Key: {
        "User ID": JSON.parse(event.body).user,
      },
      TableName: "Raw",
      UpdateExpression: addExpression,
      ExpressionAttributeNames: expressionNames,
      ExpressionAttributeValues: valueMap,
    };

    body = await dynamo.update(params).promise();
    statusCode = 200;
  } catch (error) {
    console.error('Error:', error);
    if (error.name == 'ValidationException') {
      const params = {
        Key: {
          "User ID": JSON.parse(event.body).user,
        },
        TableName: "Raw",
        UpdateExpression: "SET #Data = :d, #Timestamp =
        ExpressionAttributeNames: {
          "#Data" : "Data",
          "#Timestamp" : "Timestamp"
        },
        ExpressionAttributeValues: {
          ":d" : data,
          ":t" : new Date().toString()
        },
```

Lambda - updateUser

- Simple update functionality
- Creates new records if user does not exist
- Can take in N value propositions

```
result = {};
try {
    const params = {
        TableName: "Processed",
        KeyConditionExpression: "#UserID = :UserID",
        ExpressionAttributeNames: {
            "#UserID": "User ID"
        },
        ExpressionAttributeValues: {
            ":UserID": event.queryStringParameters
        },
        Limit: event.queryStringParameters.limit
    };

    result = await dynamo.query(params).promise();

    statusCode = 200;
    body = JSON.stringify(result);
} catch (error) {
    if (error.statusCode === 403) {
        statusCode = 403;
    } else if (error.statusCode === 404) {
        statusCode = 404;
    } else if (error.statusCode === 500){
        statusCode = 500;
    }
    body = error.message;
} finally {
    console.log({
        statusCode,
```

Lambda - getUserData

- Simple retrieve functionality
- Has a MB limit imposed by DynamoDB
- Client can run another query to retrieve remaining records

Thanks for listening!