Serverless application security

Rob Sutter – AWS Serverless
Twitch: /robsutter
Twitter: @rts_rob

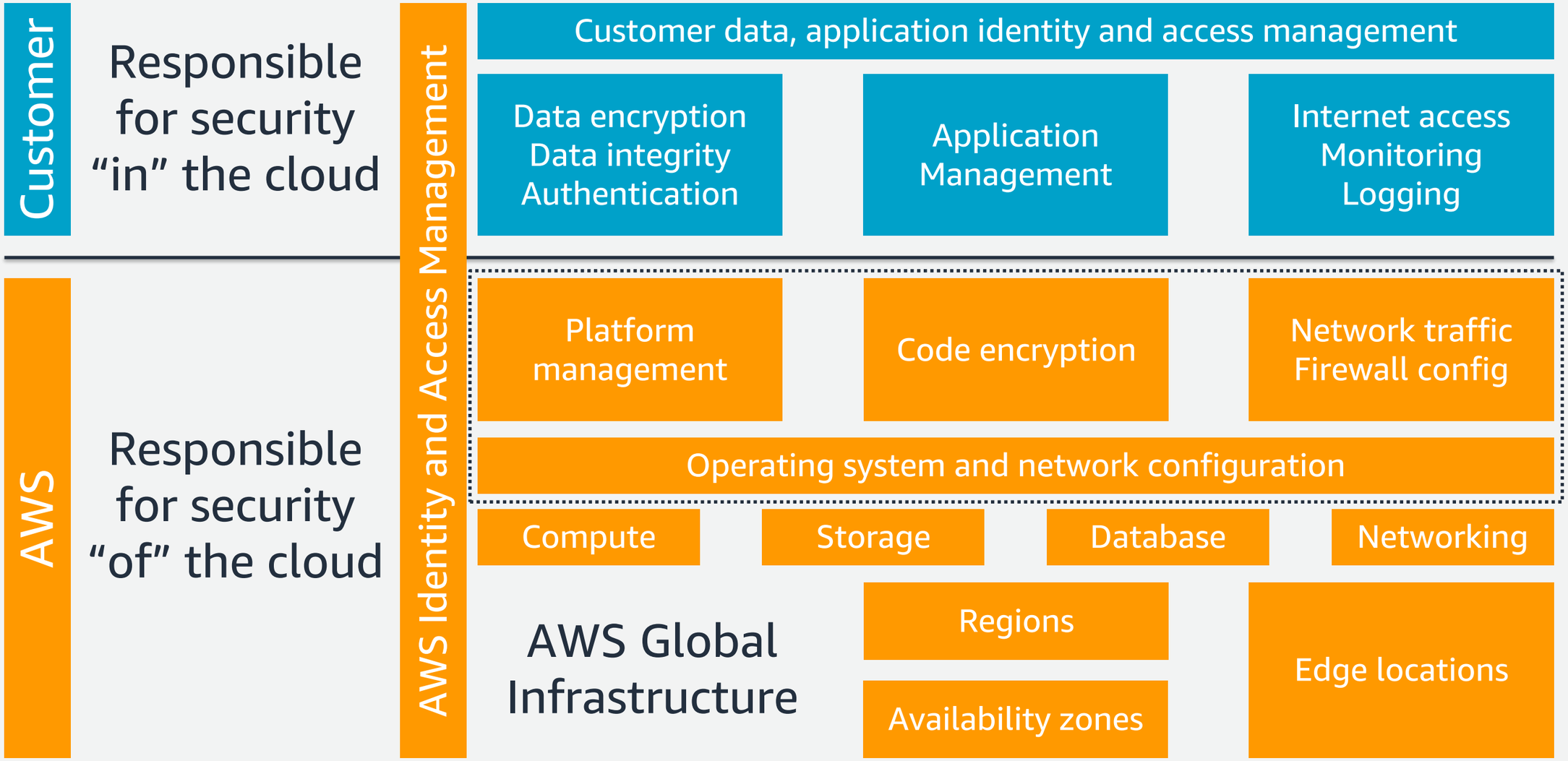
Session agenda

- How is serverless application security different?
- Similarities to traditional application security
- Service-specific security resources
- Applying security principles to Fresh Tracks

Differences

Serverless application security

AWS Shared Responsibility Model



AWS Shared Responsibility Model

AWS assumes responsibility for these components of serverless applications

Finer-grained control gives you better security

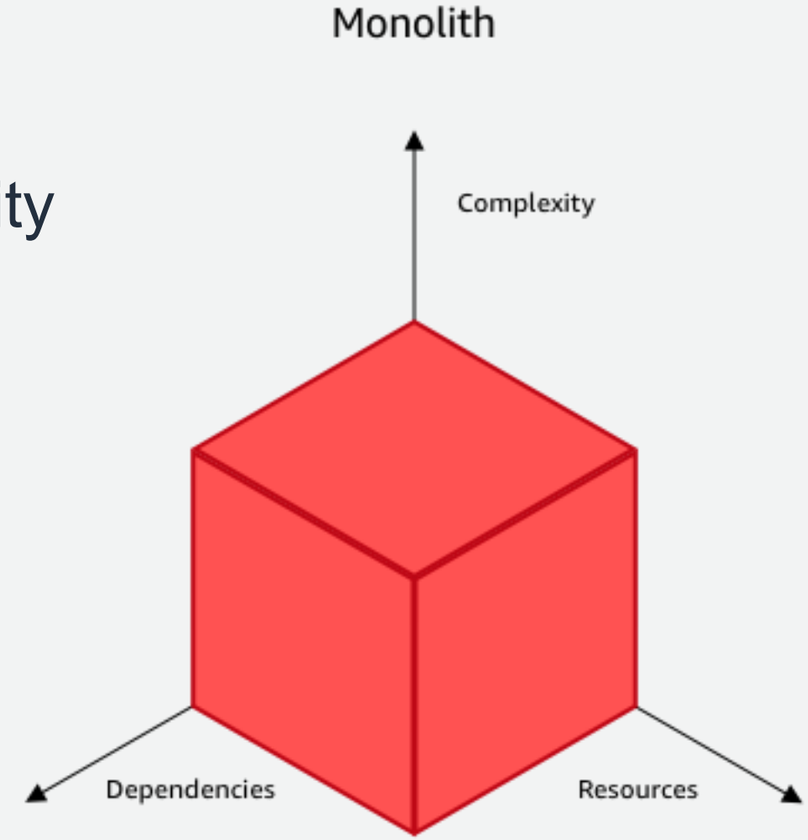
In a monolithic application (even in a container!), every line of code is exposed to every vulnerability in every dependency and has access to every resource.

Attack surface area = $\Sigma(c_f) * \Sigma(d_f)$ where:

- c_f = each function's computational complexity
- d_f = each function's dependencies

Potential impact = a * r where:

- a = attack surface area (see above)
- r = total number of accessible resources

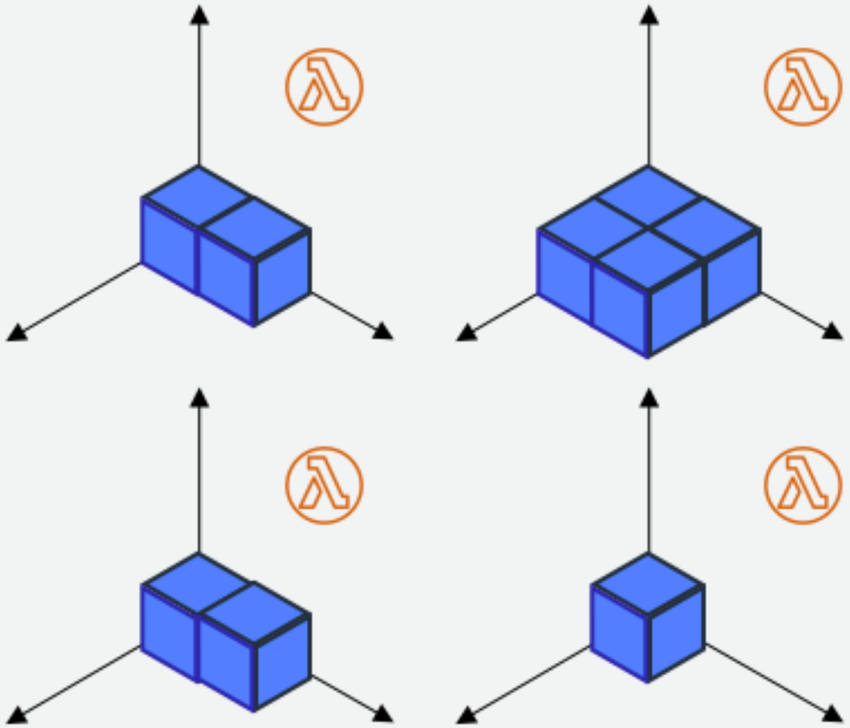


Finer-grained control gives you better security

In a well-architected serverless application, each unit of code is exposed only to the vulnerabilities in its specific logic and dependencies, and has access only to its own resources.

Potential impact = $\Sigma(c_f * d_f * r_f)$ where:

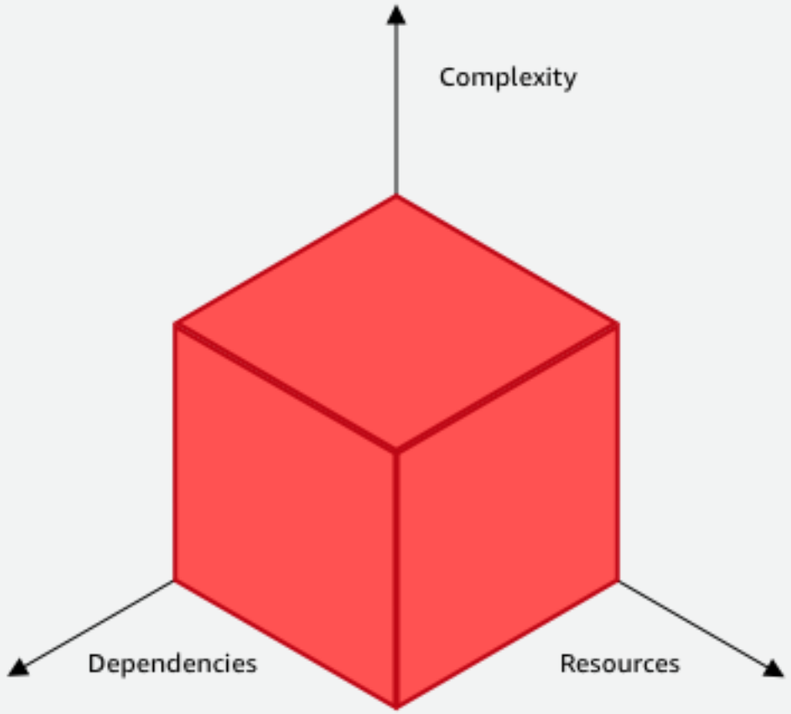
- c_f = each function's complexity
- d_f = each function's dependencies
- r_f = each function's resources



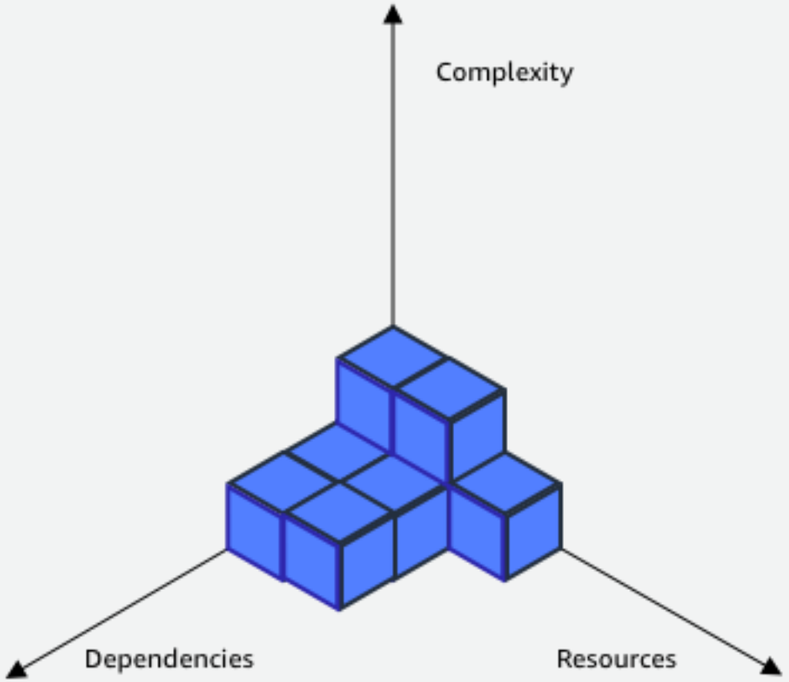
Finer-grained control gives you better security

In plain language, the potential security risk of a serverless application is lower, but still present!

Monolith



Serverless application



Similarities

Serverless application security

Serverless application security similarities

Security is not “free” with serverless. It still takes work!

- Application layer security
- Authentication and authorization
- Data encryption and integrity
- Monitoring and logging

Customer

Responsible for security “in” the cloud

Customer data, application identity and access management

Data encryption
Data integrity
Authentication

Application Management

Internet access
Monitoring
Logging

OWASP Serverless Top Ten

S1:2017 Injection

S2:2017 Broken Authentication

S3:2017 Sensitive Data Exposure

S4:2017 XML External Entities (XXE)

S5:2017 Broken Access Control

S6:2017 Security Misconfiguration

S7:2017 Cross-Site Scripting (XSS)

S8:2017 Insecure Deserialization

S9:2017 Using Components with Known Vulnerabilities

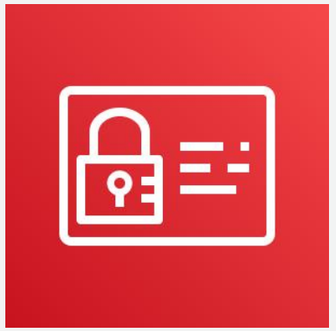
S10:2017 Insufficient Logging and Monitoring

Application layer security (S1, S3, S4, S5, S6, S7, S8:2017)

- Applications have different use cases and risk tolerances
- AWS empowers customers to build according to their needs
- A security vulnerability in one application can be indistinguishable from a critical feature in another
    - Example: a B2C platform startup enables cross-origin resource sharing (CORS) globally, whereas a financial institution restricts it entirely

Authentication and authorization (S2, S5, S6:2017)

- Use available tooling
- Amazon offers Amazon Cognito
- Partners such as Auth0
- Don’t write your own!
- AWS Identity and Access Management (IAM) ties all the pieces together



AWS Identity and Access Management

Data encryption and integrity – S3:2017

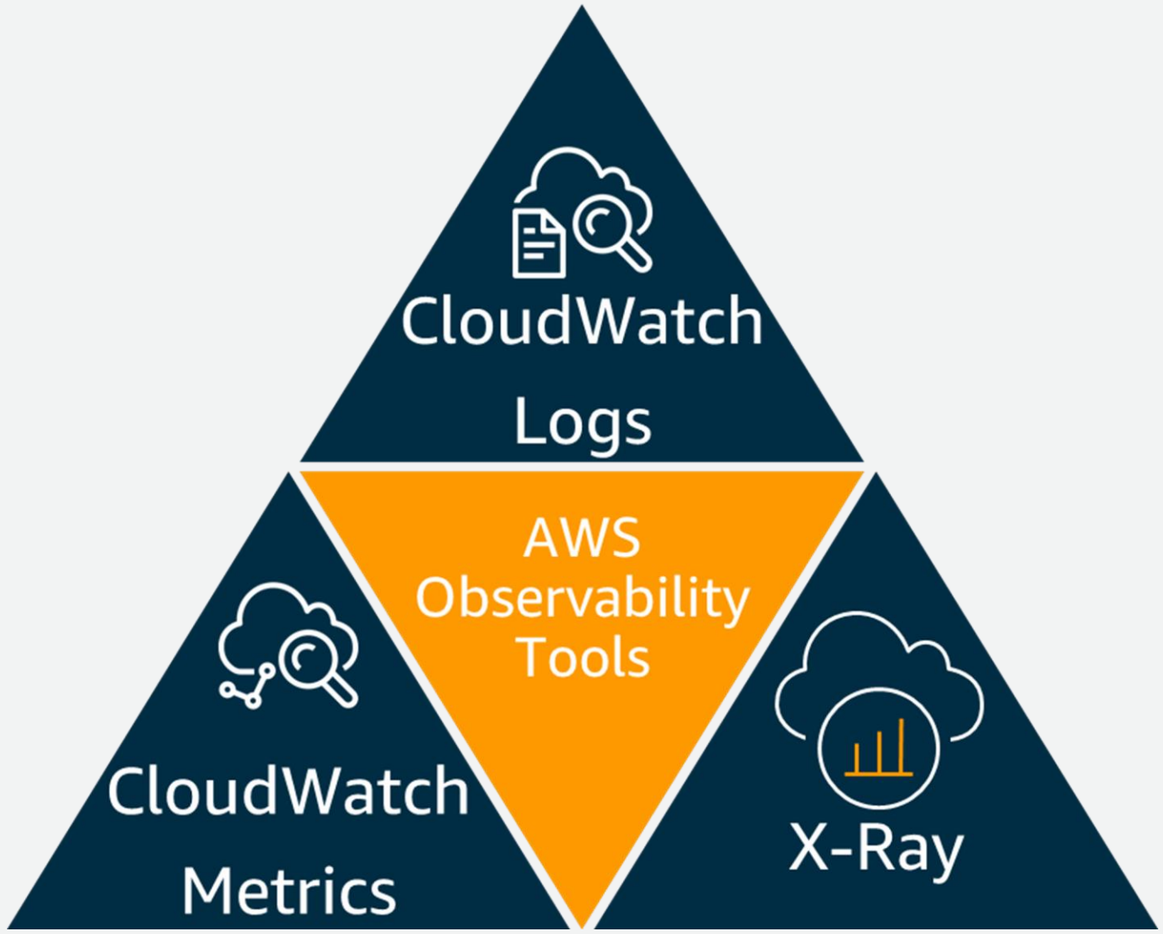
- Identify and classify sensitive data
- Minimize storage of sensitive data to only what is absolutely necessary
- Protect data at rest
- Use infrastructure provider services for key management and encryption of stored data, secrets, and environment variables

AWS Key Management Service

AWS Secrets Manager

Monitoring and logging (S10:2017)

- Use monitoring tools provided by the service provider to identify and report unwanted behavior
    - Wrong credentials
    - Unauthorized access to resources
    - Excessive execution of functions
    - Unusually long execution time



Amazon Partner Network

Aqua Security

- Dev-to-prod security across your entire CI/CD pipeline and runtime environments
- www.aquasec.com

Snyk

- Proactively finds and fixes vulnerabilities and license violations in open source dependencies
- www.snyk.io

Service-specific resources

Serverless application security

Exploring a traditional web application technology stack

AWS Amplify Console

The AWS Amplify Console properly configures an S3 bucket and Amazon CloudFront distribution for you, and can configure authentication for your app.

The key focus for customers is restricting deployments with AWS IAM.

- *CreateBranch*, *CreateDeployment*, *CreateWebHook*
- *DeleteApp*, *DeleteBranch*, *DeleteWebHook*
- *StartDeployment*, *StartJob*
- *StopJob*
- *UpdateWebHook*

AWS Amplify

Exploring a traditional web application technology stack

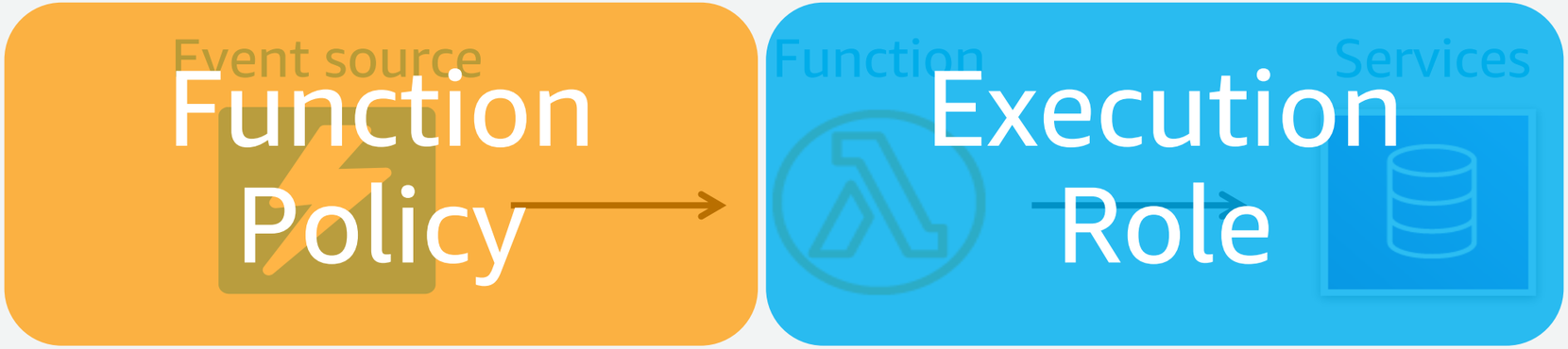
AWS Lambda

Function policies:

- “Actions on bucket X can invoke Lambda function Z"
- Resource policies allow for cross account access
- Used for sync and async invocations

Execution role:

- “Lambda function A can read from DynamoDB table users”
- Define what AWS resources/API calls can this function access via IAM
- Used in streaming invocations



AWS Lambda – Function policy

Created implicitly by AWS SAM when you attach events.

The SAM template shown here allows Amazon API Gateway to invoke the *saveToFreshTracksDatabaseTable* Lambda function

```
saveToFreshTracksDatabaseTable:
  Type: AWS::Serverless::Function
  Properties:
    Events:
      HttpPost:
        Type: Api
        Properties:
          Auth:
            Authorizer: MyLambdaTokenAuthorizer
          Path: '/activity'
          Method: post
          RestApiId: !Ref FreshTracksAPI
```

AWS Lambda – Execution role

Created explicitly by you when you define your function.

The SAM template shown here allows the *saveToFreshTracksDatabaseTable* Lambda function to read from and write to the *FreshTracksDatabaseTabl* Amazon DynamoDB table.

```
saveToFreshTracksDatabaseTable:
  Type: AWS::Serverless::Function
  Properties:
    Policies:
      - DynamoDBCrudPolicy:
          TableName: !Ref FreshTracksDatabaseTable
```

AWS SAM policy templates

- Included in the AWS Serverless Application Model (SAM)
- Help you quickly scope permissions to the resources used by your application
- Applications that use policy templates don’t require acknowledgements to deploy from the AWS Serverless Application Repository
- Open Source: submit pull requests and issues at:
    - github.com/awslabs/serverless-application-model/

AWS SAM policy templates

For more information and a complete list see: rbsttr.tv/sampolicy

```yaml
MyFunction:
  Type: AWS::Serverless::Function
  Properties:
    Policies:
      - SQSPollerPolicy: # Policy template with placeholder value
          QueueName:
            !GetAtt MyQueue.QueueName
      - CloudWatchPutMetricPolicy: {}  # Policy template with no placeholder value
      - DynamoDBWritePolicy:
          TableName:
            !Ref MyTable
```

AWS SAM policy templates

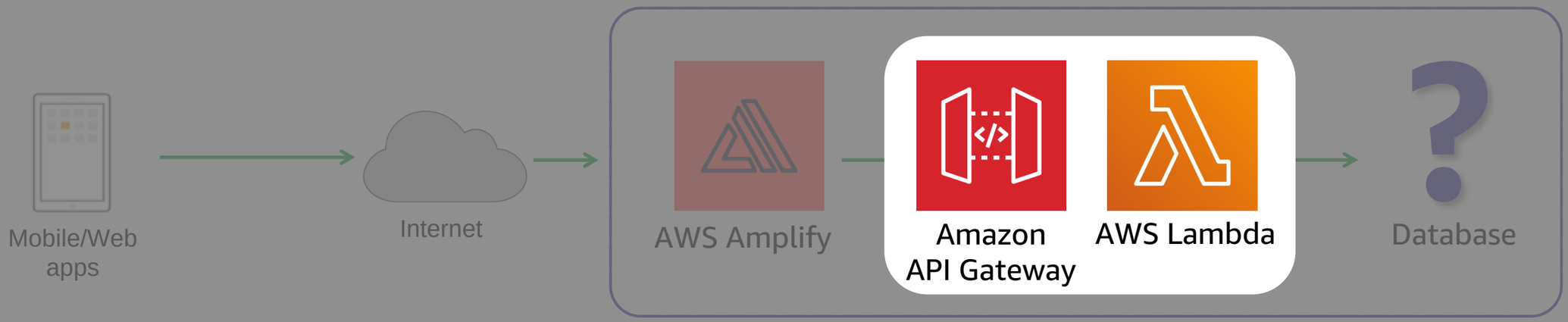
These two lines:

```
- DynamoDBReadPolicy:
    TableName: !Ref MyTable
```

Become this complete policy:

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "dynamodb:GetItem",
      "dynamodb:Scan",
      "dynamodb:Query",
      "dynamodb:BatchGetItem",
      "dynamodb:DescribeTable"
    ],
    "Resource": [
      {
        "Fn::Sub": [
          "arn:${AWS::Partition}:dynamodb:${AWS::Region}:${AWS::AccountId}:table/${tableName}",
          {
            "tableName": {
              "Ref": "TableName"
            }
          }
        ]
      },
      {
        "Fn::Sub": [
          "arn:${AWS::Partition}:dynamodb:${AWS::Region}:${AWS::AccountId}:table/${tableName}/index/*",
          {
            "tableName": {
              "Ref": "TableName"
            }
          }
        ]
      }
    ]
  }
]
```

Exploring a traditional web application technology stack



Amazon API Gateway

IAM permissions

- Use IAM policies and AWS credentials to grant access

Lambda Authorizers

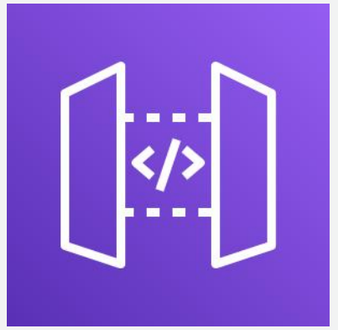
- Use a Lambda function to validate a bearer token, e.g., OAuth or SAML

Cognito User Pools

- Create a completely managed user management system

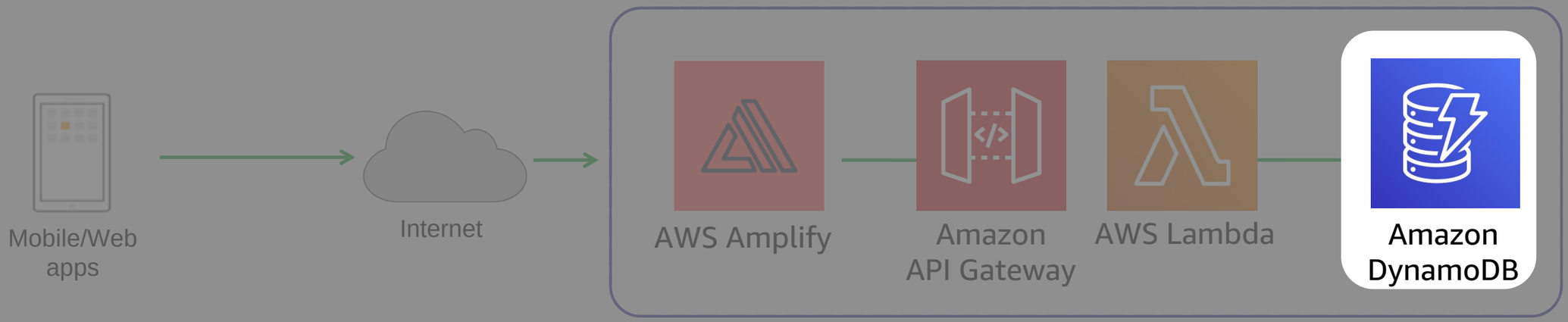
Resource Policies

- Can restrict based on IP, VPC, AWS Account ID



Amazon API Gateway

Exploring a traditional web application technology stack



Amazon DynamoDB

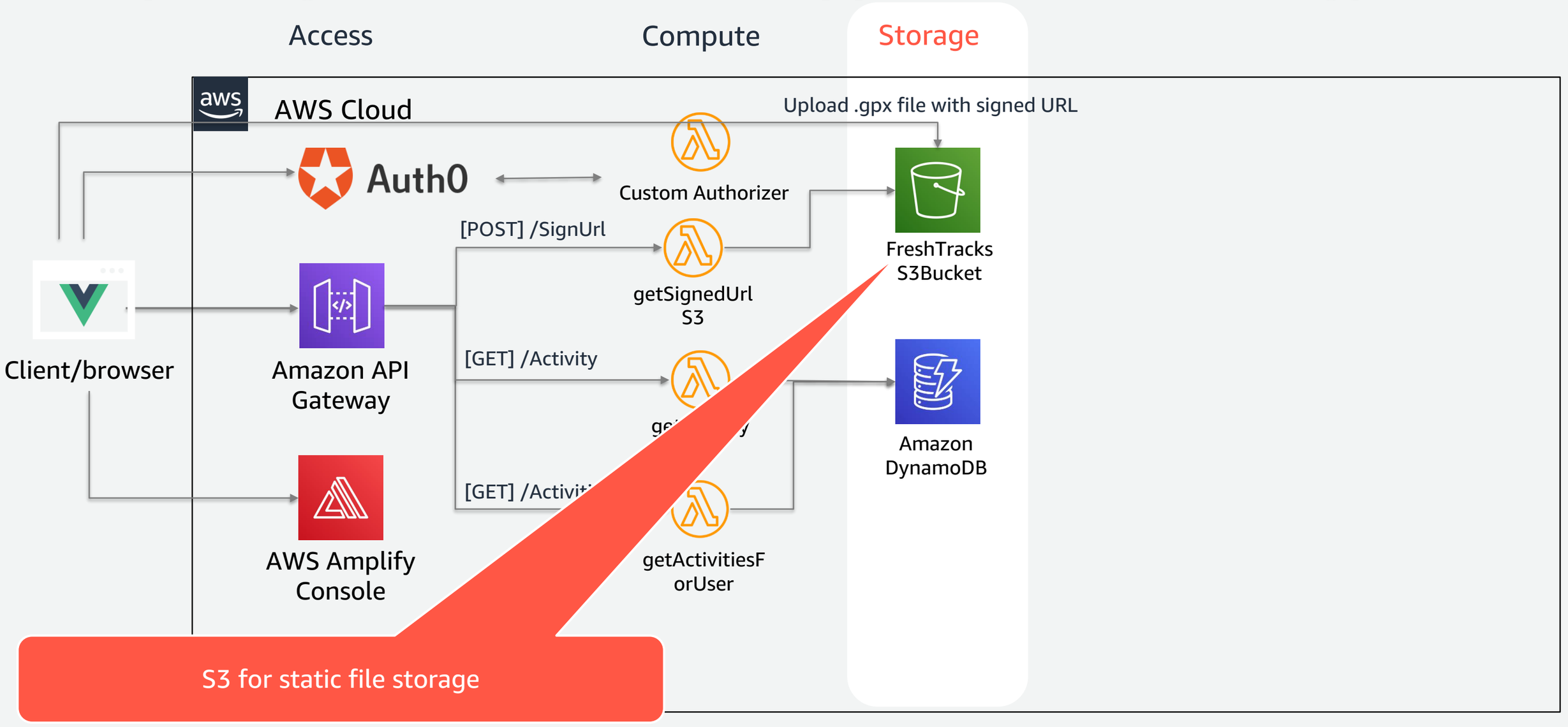
Start with the AWS SAM policy templates:

- *DynamoDBReadPolicy* for read-only
- *DynamoDBWritePolicy* for creates and updates
- *DynamoDBStreamReadPolicy* to attach to streams
- Avoid *DynamoDBCrudPolicy* whenever possible
- Command-query responsibility separation (CQRS)

Amazon DynamoDB

Allows for extremely fine-grained access via the IAM condition *dynamodb:LeadingKeys*

Exploring a serverless web application technology stack



Amazon S3

S3 buckets are not public by default
In general you **should not change this!**

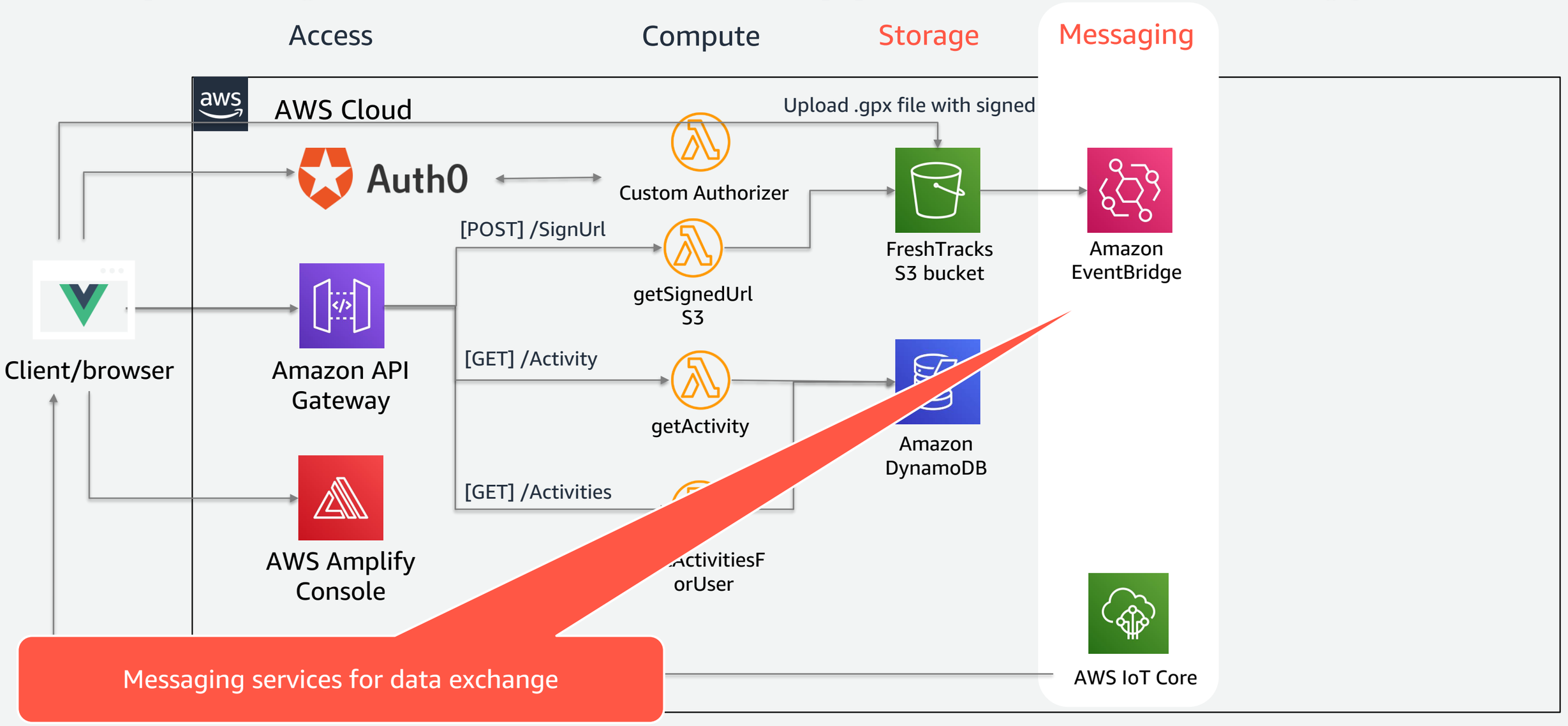
Amazon Simple Storage Service

Again, take advantage of AWS SAM policy templates:

- *S3ReadPolicy* for retrieving data
- *S3WritePolicy* for storing data
- Avoid using *S3CrudPolicy* and *S3FullAccessPolicy* whenever possible

Use S3 Access Points for even greater control over access to your buckets

Exploring a serverless web application technology stack



Amazon EventBridge

AWS IAM offers permissions for inbound and outbound operations

Inbound operations determine what principals can place events onto event buses and define rules and targets:

- *events:PutEvents*
- *events:PutRule*
- *events:PutTargets*

Lambda function

Custom event bus

```
PublishEventsFunction:
  Type: AWS::Serverless::Function
  Properties:
    CodeUri: publish-events/
    Handler: app.lambdaHandler
    Runtime: nodejs12.x
    Policies:
      - Statement:
          - Effect: Allow
            Action:
              - events:PutEvents
            Resource:
              - !GetAtt EventBus.Arn
```

Amazon EventBridge

AWS IAM offers permissions for inbound and outbound operations

Outbound permissions are determined by the receiving resource.

Amazon EventBridge

AWS Express Workflows

```yaml
InvokeWorkflowRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: "2012-10-17"
      Statement:
        - Effect: Allow
          Principal:
            Service:
              - events.amazonaws.com
          Action: sts:AssumeRole
    Policies:
      - PolicyName: InvokeCustomerWorkflowsPolicy
        PolicyDocument:
          Version: "2012-10-17"
          Statement:
            - Effect: Allow
              Action:
                - states:StartExecution
              Resource:
                - !Ref MyExpressWorkflow
```

AWS IoT Core

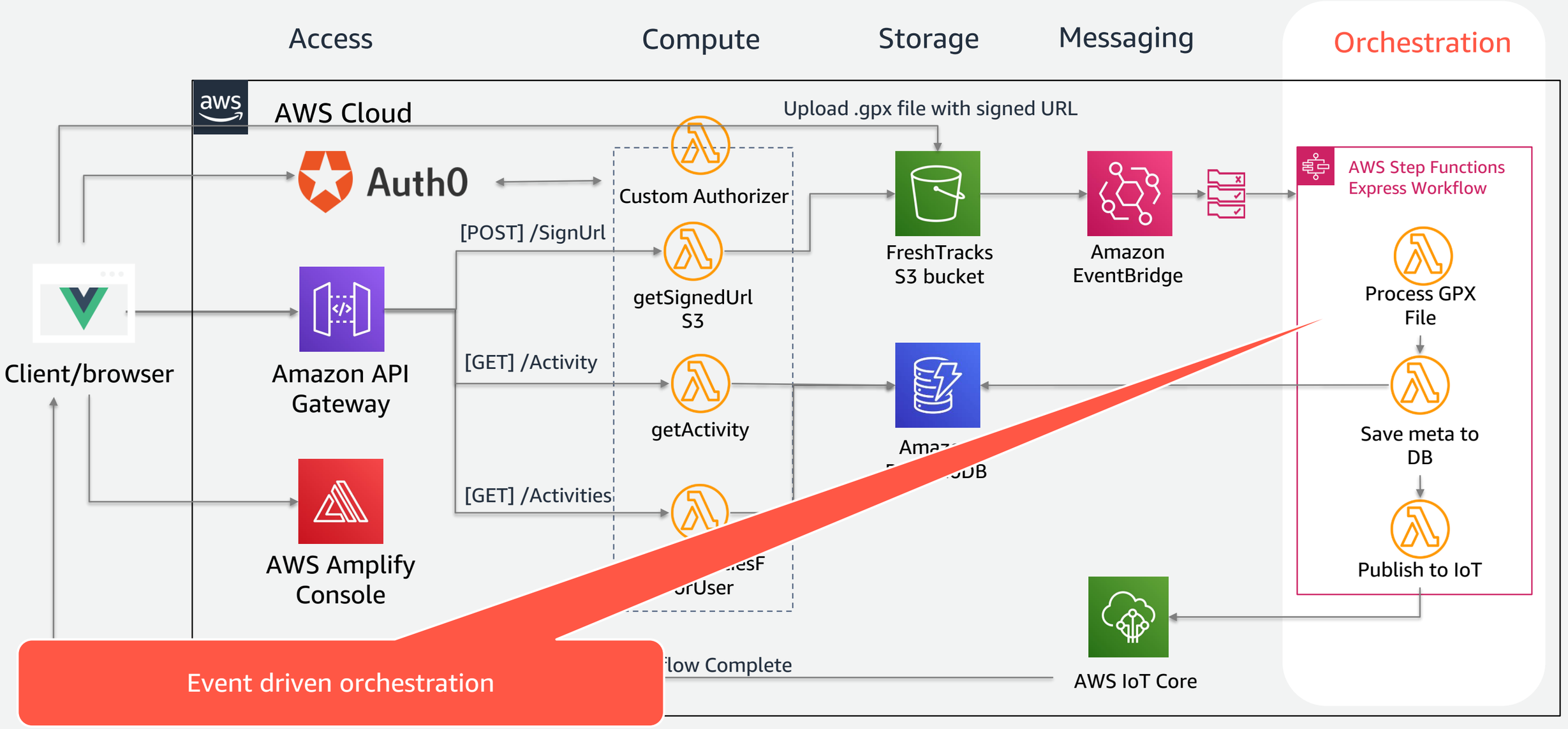
AWS IAM policies and AWS IoT Core policies

AWS IAM also provides a set of IAM managed policies

- *AWSIoTDataAccess*
- *AWSIoTEventsReadOnlyAccess*
- *AWSIoTLogging*

For more information and a complete list see: rbsttr.tv/iotiam

Exploring a serverless web application technology stack

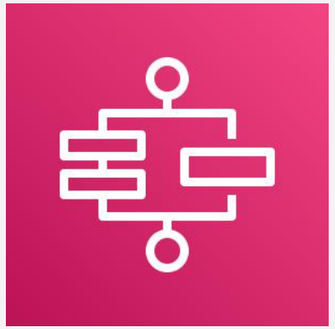


AWS Step Functions

- Data in AWS Step Functions is encrypted at rest
- All data that passes between Step Functions and integrated services is encrypted using Transport Layer Security (TLS)

AWS IAM governs Step Functions executions and invocations

- Special consideration for service integrations
    - Run a Job (*.sync*)
    - Wait for Callback (*.waitForTaskToken*)



Standard Workflows

Express Workflows

Compliance

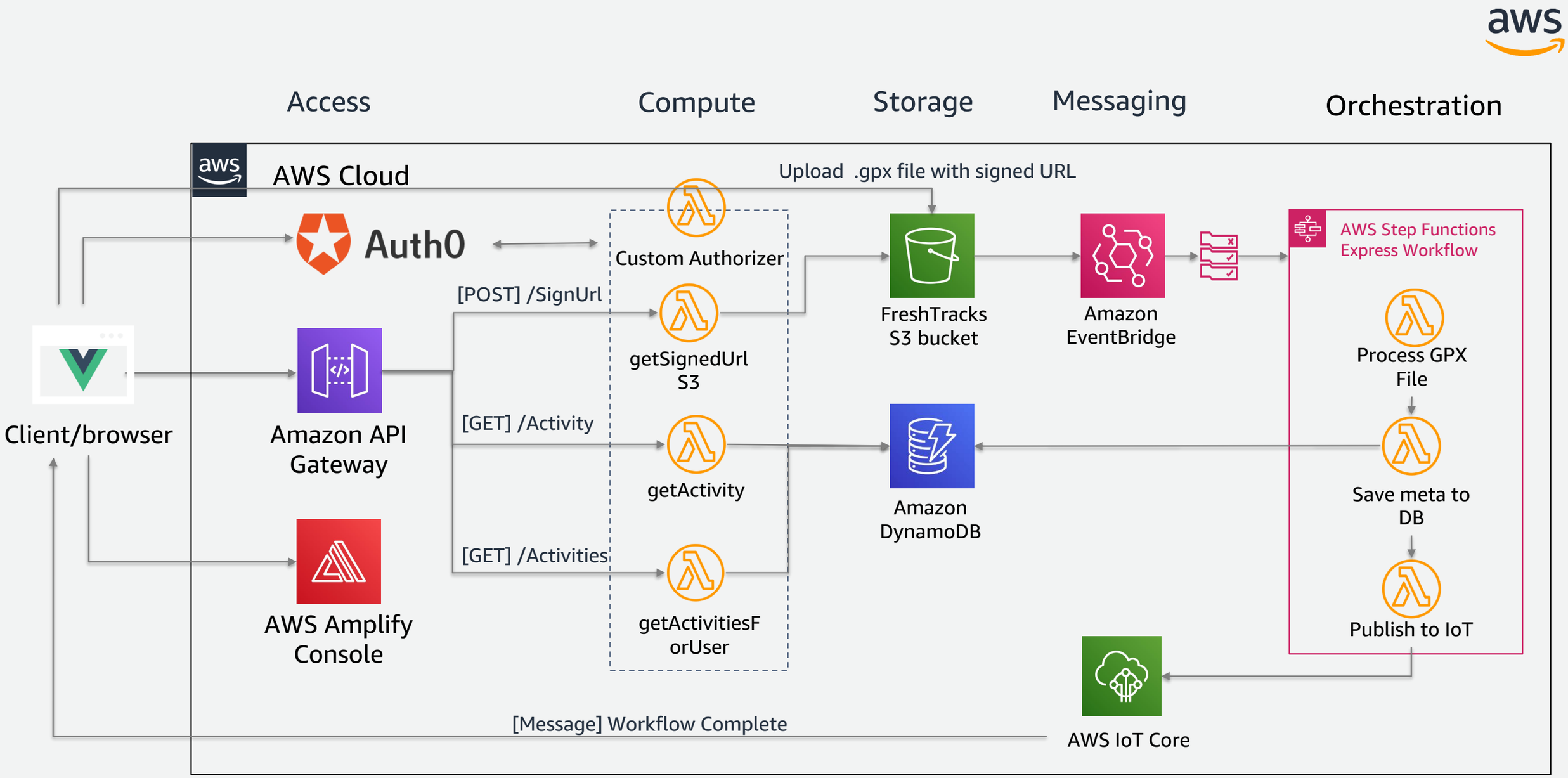
- Compliance-ready for SOC, PCI, FedRAMP, HIPAA, and others

Service	SOC	PCI	ISO	FedRAMP	HIPAA
AWS Amplify Console	✓	✓	✓		✓
AWS Lambda	✓	✓	✓	✓	✓
Amazon API Gateway	✓	✓	✓	✓	✓
Amazon DynamoDB	✓	✓	✓	✓	✓
Amazon S3	✓	✓	✓	✓	✓
Amazon EventBridge	✓	✓	✓	✓	✓
AWS IoT Core	✓	✓	✓	✓	✓
AWS Step Functions	✓	✓	✓	✓	✓

Learn more at https://aws.amazon.com/compliance/services-in-scope/

Securing Fresh Tracks

Serverless application security



Optimization best practices are also security best practices

Avoid monolithic functions

- Reduces complexity
- Reduces number of resources
- Both reduce potential impact

Optimize dependencies (and imports)

- Reduces complexity
- Reduces the attack surface

AWS Lambda

Function	Policies
CreateZendeskArticle	AWSLambdaBasicExecutionRole
GetFullZendeskTicket	AWSLambdaBasicExecutionRole
GetFullZendeskUser	AWSLambdaBasicExecutionRole
publishToIoT	Inline - Action: iot:*, Resource: *
SaveAuth0EventToS3	S3CrudPolicy
saveToFreshTracksDatabaseTable	DynamoDBCrudPolicy
getActivitiesForUser	DynamoDBCrudPolicy
getActivity	DynamoDBCrudPolicy, S3CrudPolicy
parseGPX	DynamoDBCrudPolicy, S3CrudPolicy
getSignedUrlS3	S3CrudPolicy

AWS Lambda

Our *publishToIoT* function uses an overly broad inline policy.

How can we improve this?

```yaml
publishToIoT:
  Type: AWS::Serverless::Function
  Properties:
    CodeUri: Lambda/
    Handler: publishToIoT.handler
    Runtime: nodejs12.x
    MemorySize: 128
    Environment:
      Variables:
        Endpoint: !Ref FreshTracksRealtime
    Policies:
      - Statement:
        - Effect: Allow
          Action:
            - "iot:*"
          Resource:
            - "*"
        Version: '2012-10-17'
```

```javascript
const AWS = require('aws-sdk')
const iot = new AWS.Iot({apiVersion: '2015-05-28'});

exports.handler = async(event, context) => {

    var params = {endpointType: 'iot:Data'};
    const iotDataRes = await   iot.describeEndpoint(params).promise();
    const iotdata = new AWS.IotData({endpoint:iotDataRes.endpointAddress})
    const res = await iotdata.publish(...).promise();

    return res

}
```

AWS Lambda

We have two IOT API calls in our code: *describeEndpoint* and *publish*

- *describeEndpoint* does not take any Resource arguments
- *publish* accepts the ARN of an IoT topic as a Resource argument
- *FreshTracksRealtime* is the IoT topic defined in our SAM template
- We use *!GetAtt* to obtain the ARN of the topic

AWS Lambda

Now our function is restricted to:

- only the API calls it needs to execute successfully (*describeEndpoint* and *publish*)
- only performing those API calls against the required resources (the *FreshTracksRealtime* IoT topic)

AWS SAM per-function IAM roles enable tight scoping of permissions.

```
publishToIoT:
  Type: AWS::Serverless::Function
    ...
    Policies:
      - Statement:
        - Effect: Allow
          Action:
            - "iot:describeEndpoint"
          Resource:
            - "*"
        - Effect: Allow
          Action:
            - "iot:publish"
          Resource:
            - !GetAtt FreshTracksRealtime.Arn
        Version: '2012-10-17'
```

Amazon API Gateway

Our API Gateway CORS policy is open to the world.

How can we improve this?

```
Globals:
  Function:
    Timeout: 10
  Api:
    # enable CORS; to make more specific, change the origin wildcard
    # to a particular domain name, e.g. "'www.example.com'"
    EndpointConfiguration: EDGE
    Cors:
      AllowMethods: "'OPTIONS,POST'"
      AllowHeaders: "'Content-Type'"
      AllowOrigin: "'*'"
```

Amazon API Gateway

Our domain name is *myfreshtracks.com*

We can instruct API Gateway to only allow traffic originating from our domain.

```
Globals:
  Function:
    Timeout: 10
  Api:
    # enable CORS; to make more specific, change the origin wildcard
    # to a particular domain name, e.g. "'www.example.com'"
    EndpointConfiguration: EDGE
    Cors:
      AllowMethods: "'OPTIONS,POST'"
      AllowHeaders: "'Content-Type'"
      AllowOrigin: "'myfreshtracks.com'"
```

Amazon API Gateway

We also enable a custom authorizer to restrict traffic to protected routes.

A custom authorizer is a Lambda function that inspects claims in a token and determines whether to permit or reject the request.

```
FreshTracksAPI:
  Type: AWS::Serverless::Api
  Properties:
    StageName: Prod
    Auth:
      Authorizers:
        MyLambdaTokenAuthorizer:
          FunctionArn: !GetAtt MyAuthFunction.Arn
```

Amazon DynamoDB

We have four functions that access our DynamoDB table.

They all use the *DynamoDBCrudPolicy*. How can we improve this?

Function	Policies
saveToFreshTracksDatabaseTable	DynamoDBCrudPolicy
getActivitiesForUser	DynamoDBCrudPolicy
getActivity	DynamoDBCrudPolicy, S3CrudPolicy
parseGPX	DynamoDBCrudPolicy, S3CrudPolicy

Amazon DynamoDB

Inspect the code for actual API calls.

Function	DynamoDB API Calls
saveToFreshTracksDatabaseTable	dynamodb.put
getActivitiesForUser	dynamodb.query
getActivity	dynamodb.getItem
parseGPX	<none>

Amazon DynamoDB

Provide the proper AWS SAM policy template

Function	API Call	Policies
saveToFreshTracksDatabaseTable	dynamodb.put	DynamoDBWritePolicy
getActivitiesForUser	dynamodb.query	DynamoDBReadPolicy
getActivity	dynamodb.getItem	DynamoDBReadPolicy
parseGPX	<none>	<none>

Amazon DynamoDB

Provide the proper AWS SAM policy template

```
saveToFreshTracksDatabaseTable:
  Type: AWS::Serverless::Function
  Properties:
    ...
    Policies:
      - DynamoDBWritePolicy:
          TableName: !Ref FreshTracksDatabaseTable

getActivitiesForUser:
  Type: AWS::Serverless::Function
  Properties:
    ...
    Policies:
      - DynamoDBReadPolicy:
          TableName: !Ref FreshTracksDatabaseTable
```

```
getActivity:
  Type: AWS::Serverless::Function
  Properties:
    ...
    Policies:
      - DynamoDBReadPolicy:
          TableName: !Ref FreshTracksDatabaseTable
      - S3ReadPolicy:
          BucketName: !Ref FreshTracksS3Bucket

parseGPX:
    Type: AWS::Serverless::Function
    Properties:
      ...
      Policies:
        - S3ReadPolicy:
            BucketName: !Ref FreshTracksS3Bucket
```

Summary

Serverless application security is:

- balanced toward the application, not the infrastructure
- more fine-grained
- not to be taken for granted!

This is only a start! AWS provides a number of solutions to secure your applications. For more, see:

https://aws.amazon.com/security/

Q&A

Rob Sutter – AWS Serverless
Twitch: /robsutter
Twitter: @rts_rob

Thank you!

Rob Sutter – AWS Serverless
Twitch: /robsutter
Twitter: @rts_rob