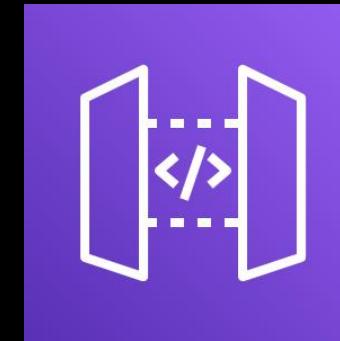




Serverless Web APIs (Contd.)



API Gateway

HTTP & REST Web APIs

Components of a Serverless app.

Logic



Lambda

State



RDS



DynamoDB



Amazon S3

Communication



API Gateway



Kinesis



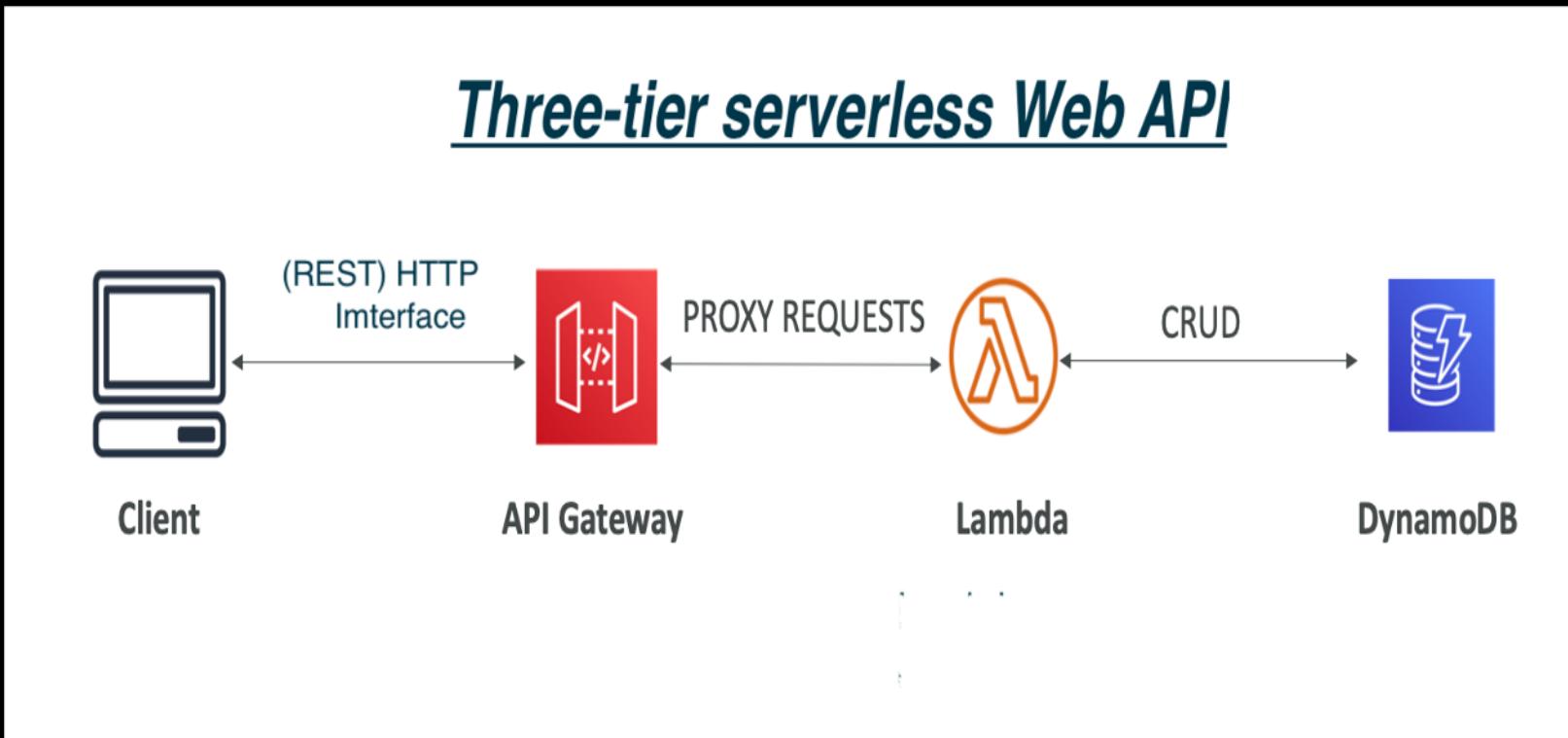
SNS



SQS

API Gateway

- A fully managed (aka Serverless) service for creating, publishing, maintaining, monitoring, and securing REST, HTTP, and WebSocket APIs.

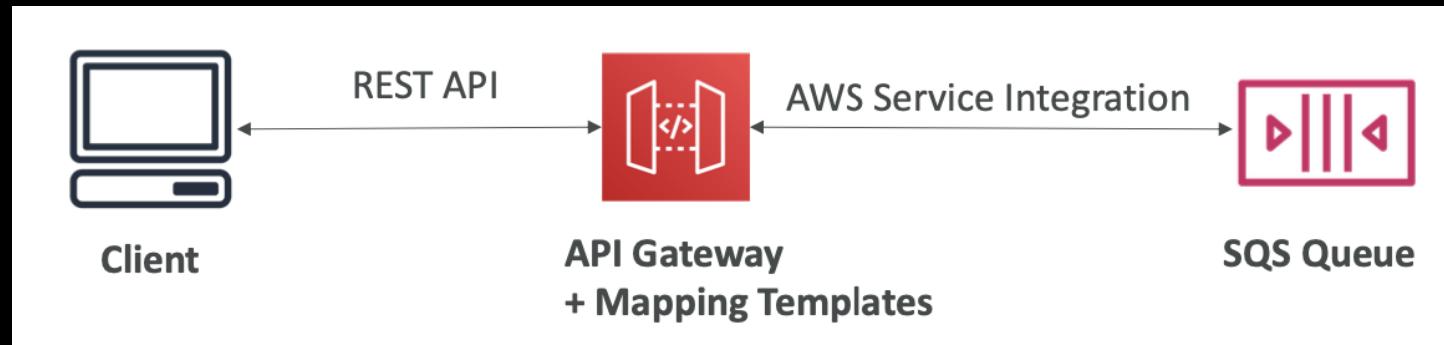


Features

- Handles API versioning (v1, v2...).
- Handles different environments (dev, test, prod).
- Handle security (Authentication and Authorization).
- Create API keys; handles request throttling.
- Swagger / Open API import to quickly define APIs.
- Support for the WebSocket Protocol.
- Transform and validate requests and responses.
- Generate SDK and API specifications.
- Caching API responses.
- DDoS protection.

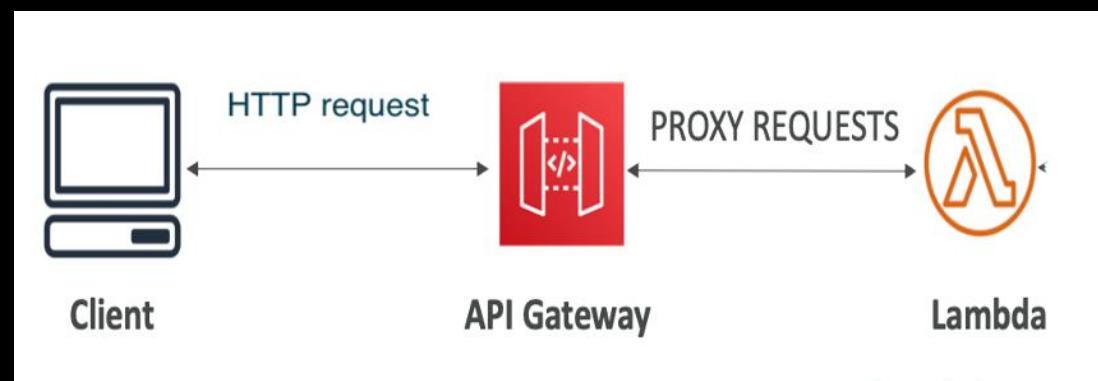
Integration Types.

- MOCK Integration Type:
 - API Gateway returns a response without sending the request to the backend.
- HTTP/AWS (AWS Services) Integration Type
 - You must configure both the integration request and response.
 - Setup data mapping using mapping templates for the request & response.



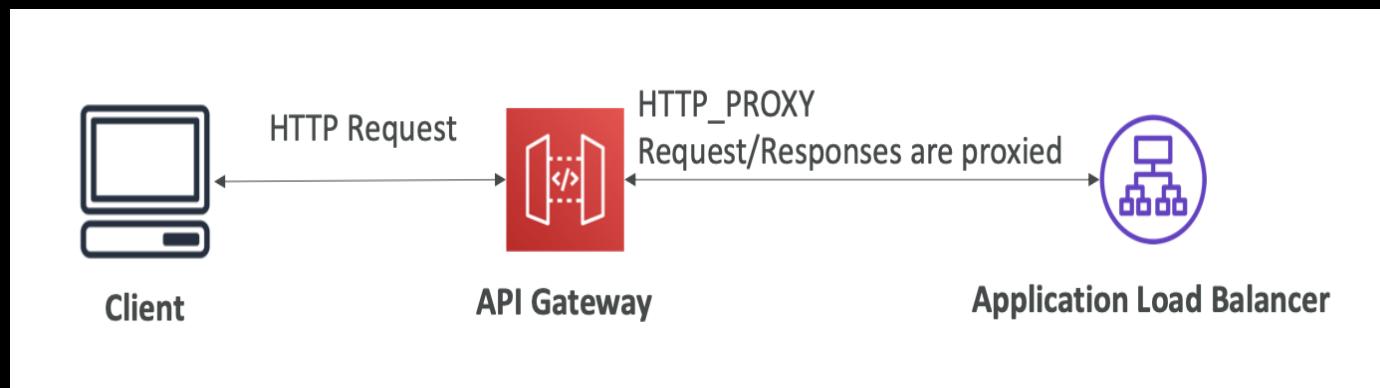
Integration Types.

- AWS_PROXY (Lambda Proxy) Integration Type:
 - Incoming request from the client is the input to Lambda.
 - No mapping template.
 - HTTP headers, query string parameters are passed as event object properties.



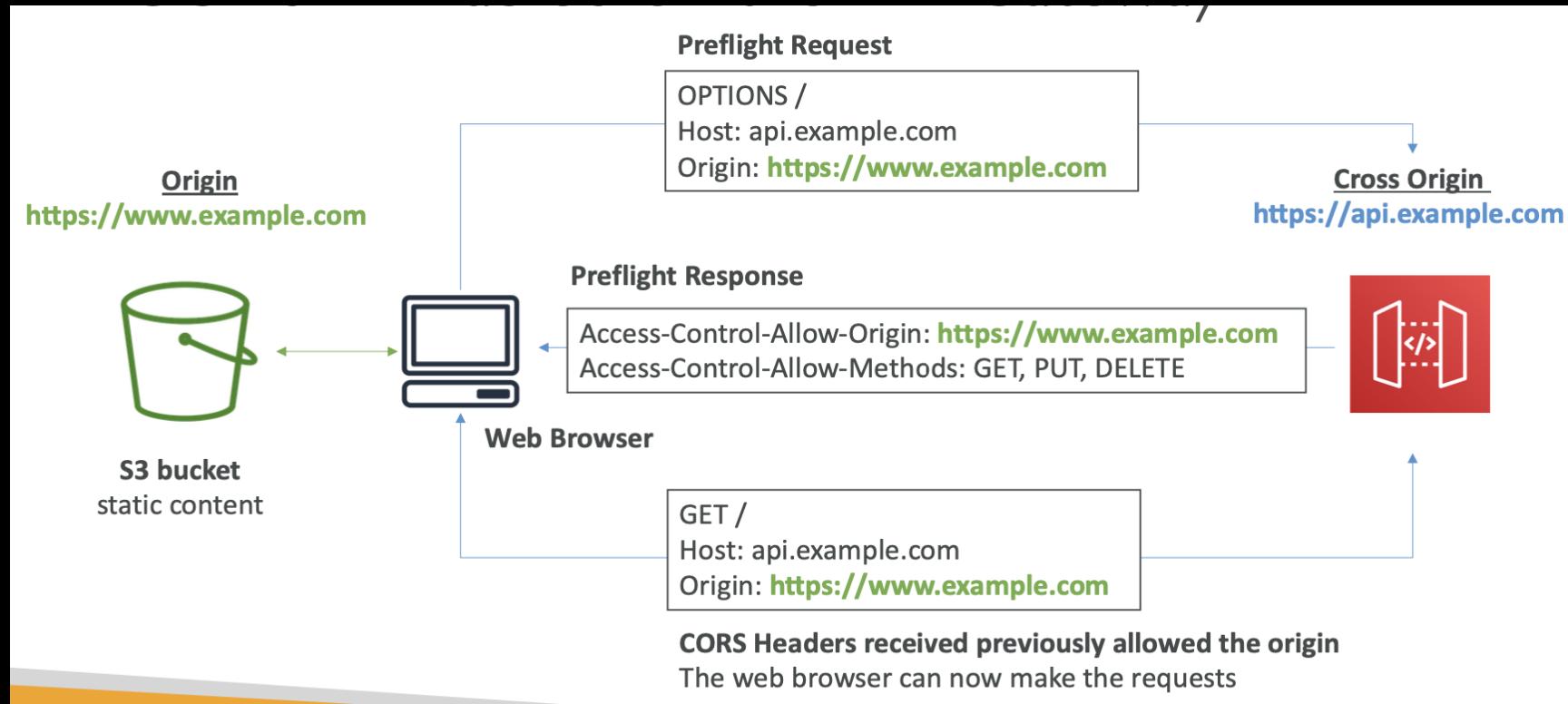
Integration Types.

- HTTP_PROXY Integration Type.
 - No mapping template.
 - The HTTP request is passed to the backend.
 - The HTTP response from the backend is forwarded by API Gateway.
 - Cheaper, more lightweight than AWS_PROXY



CORS (Cross Origin Resource Sharing)

- CORS must be enabled when you receive API calls from another domain.



Sample

- Objective – Develop a serverless Web API for managing TODOs.
 - Endpoints:
 - /todos path
 - GET – Read all todos
 - POST – Add new TODO
 - /todos/{todoID} path
 - GET – Read specific TODO

Sample – API Gateway mgt. console

The image displays two screenshots of the AWS API Gateway management console.

Top Screenshot (APIs View):

- Left Sidebar:** Shows navigation links for "APIs", "Custom domain names", and "VPC links".
- Content Area:** Title "APIs (1)". A search bar with placeholder "Find APIs". A table with columns: Name, Description, ID, Protocol. One entry: DemoAPI, example api gateway, ganf19s9t8, REST.

Bottom Screenshot (Method Execution View):

- Header:** Shows the path: Amazon API Gateway > APIs > DemoAPI (ganf19s9t8) > Resources > /todos (ycf4jb) > POST.
- Left Sidebar:** Shows "API: DemoAPI" and "Resources".
- Central Area:** A tree view of resources under "/":
 - OPTIONS
 - /todos
 - GET
 - OPTIONS
 - POST** (highlighted with a red box)
 - /todold
 - GET
 - OPTIONS
- Right Area:** Shows the flow from "Method Request" to "Integration Request".
 - Method Request:** Auth: NONE, ARN: arn:aws:execute-api:eu-west-1:517039770760:ganf19s9t8/*/P
 - Integration Request:** Type: LAMBDA_PROXY

```
// Lambda backend functionality

const RESTEndpointFn = new NodejsFunction(this, "RESTEndpointFn", ....);

// API Gateway configuration

const api = new RestApi(this, "DemoAPI", {
  description: "example api gateway",
  deployOptions: {
    stageName: "dev",
  },
  defaultCorsPreflightOptions: {
    allowMethods: ["OPTIONS", "GET", "POST", "PUT", "PATCH", "DELETE"],
    allowOrigins: ["*"],
  },
});

const todosEndpoint = api.root.addResource("todos");
todosEndpoint.addMethod(
  "GET",
  new LambdaIntegration(RESTEndpointFn, { proxy: true }) // AWSIntegration
);
todosEndpoint.addMethod(
  "POST",
  new LambdaIntegration(RESTEndpointFn, { proxy: true }) // AWSIntegration
);

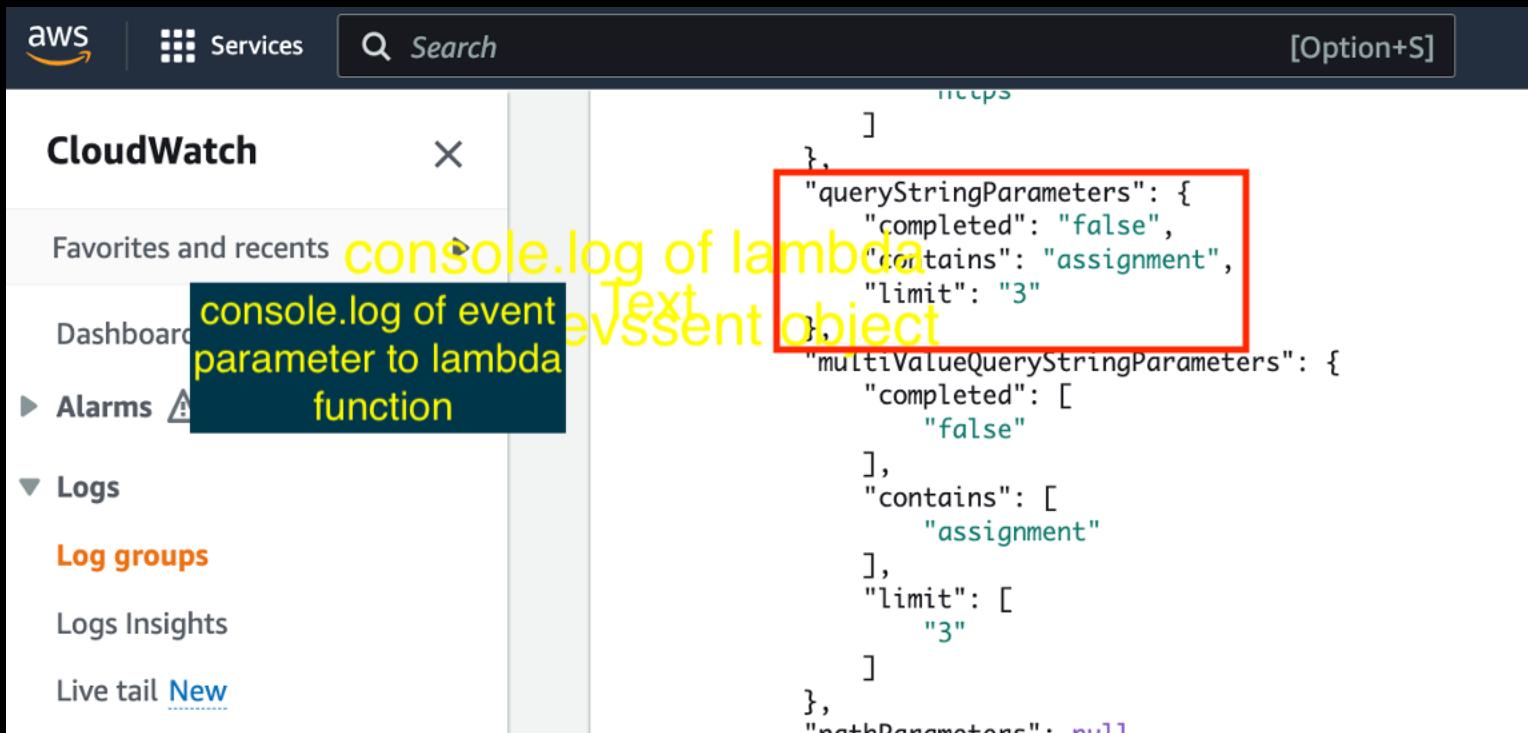
const todoDetailsEndpoint = todosEndpoint.addResource("{todoId}");
todoEndpoint.addMethod(
  "GET",
  new LambdaIntegration(RESTEndpointFn, { proxy: true })
);
```

CDK stack to create
REST Web API, using
API Gateway service

Sample

- For AWS_PROXY (Lambda Proxy) integration, the HTTP query string is parsed and added to the lambda's event argument, e.g.

https://ganf19s9t8.execute-api.eu-west-1.amazonaws.com /dev/todos?
contains=assignment&completed=false&limit=3



The screenshot shows the AWS CloudWatch Logs interface. On the left, there is a sidebar with 'CloudWatch' selected. The main area displays a log entry with yellow annotations:

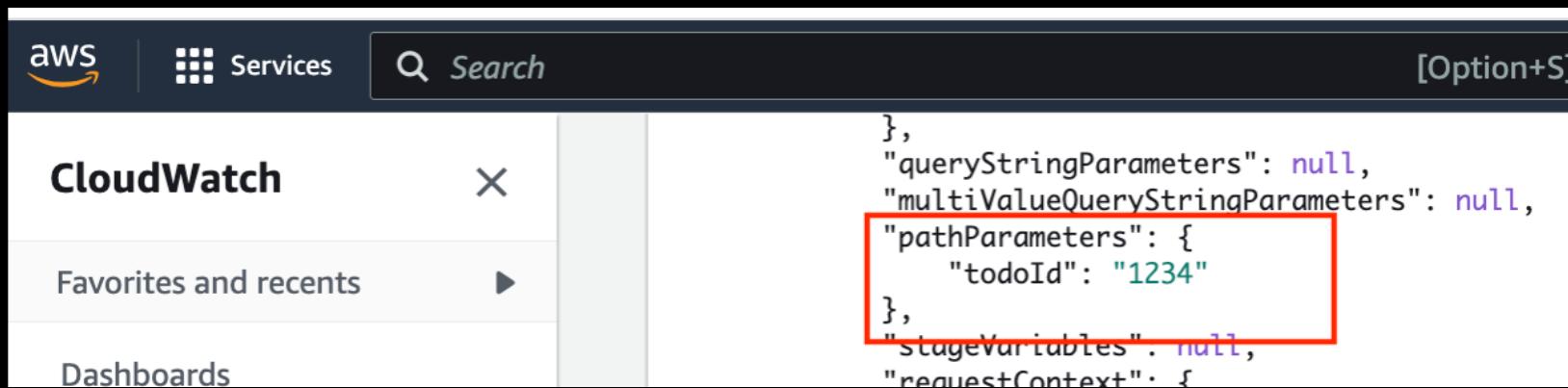
- 'console.log of event parameter to lambda function' is written over the log line.
- 'Text' is written over the word 'Text' in the log entry.
- 'vs sent object' is written over the word 'object' in the log entry.
- A red box highlights the 'queryStringParameters' object in the JSON log entry.

```
    ],
  },
  "queryStringParameters": {
    "completed": "false",
    "contains": "assignment",
    "limit": "3"
  },
  "multiValueQueryStringParameters": {
    "completed": [
      "false"
    ],
    "contains": [
      "assignment"
    ],
    "limit": [
      "3"
    ]
  },
  "pathParameters": null
```

Sample

- For AWS_PROXY (Lambda Proxy) integration, HTTP path parameters are parsed and added to the lambda's event argument, e.g.

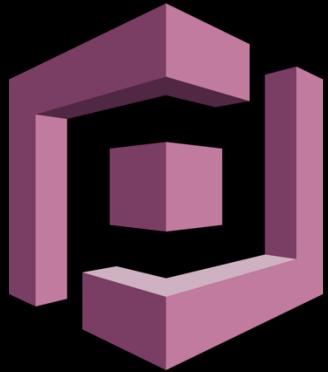
<https://ganf19s9t8.execute-api.eu-west-1.amazonaws.com/dev/todos/1234>



```
aws Services Search [Option+S]
CloudWatch Favorites and recents Dashboards
[{"id": "1", "name": "todoId", "value": "1234"}]
```

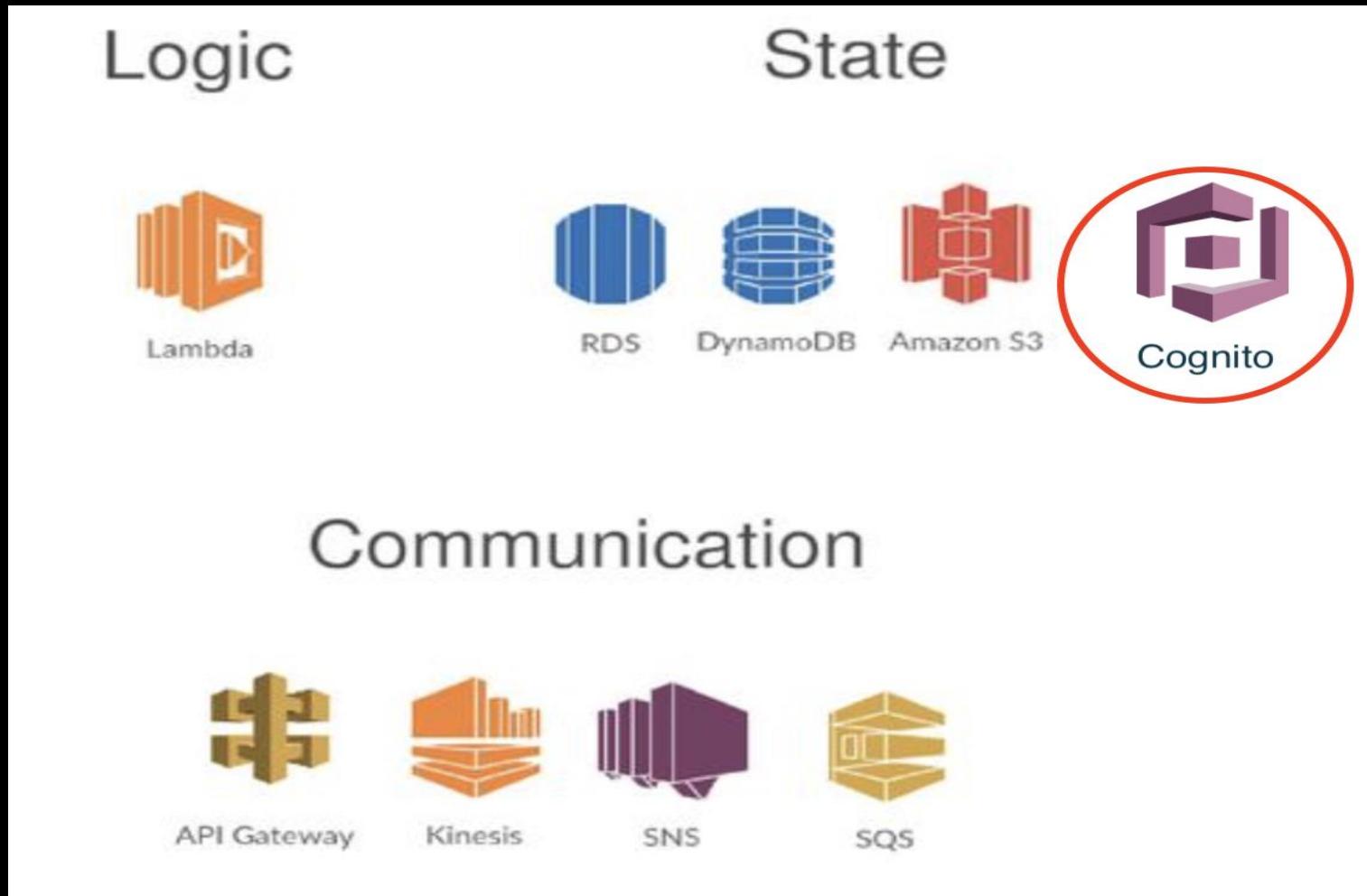
The screenshot shows the AWS CloudWatch interface with the CloudWatch service selected. On the left, there are links for 'Favorites and recents' and 'Dashboards'. On the right, a log entry is displayed with a red box highlighting the 'pathParameters' field. The log entry contains the following JSON:

```
{
  "pathParameters": {
    "todoId": "1234"
  }
}
```



Amazon Cognito

Components of a Serverless App



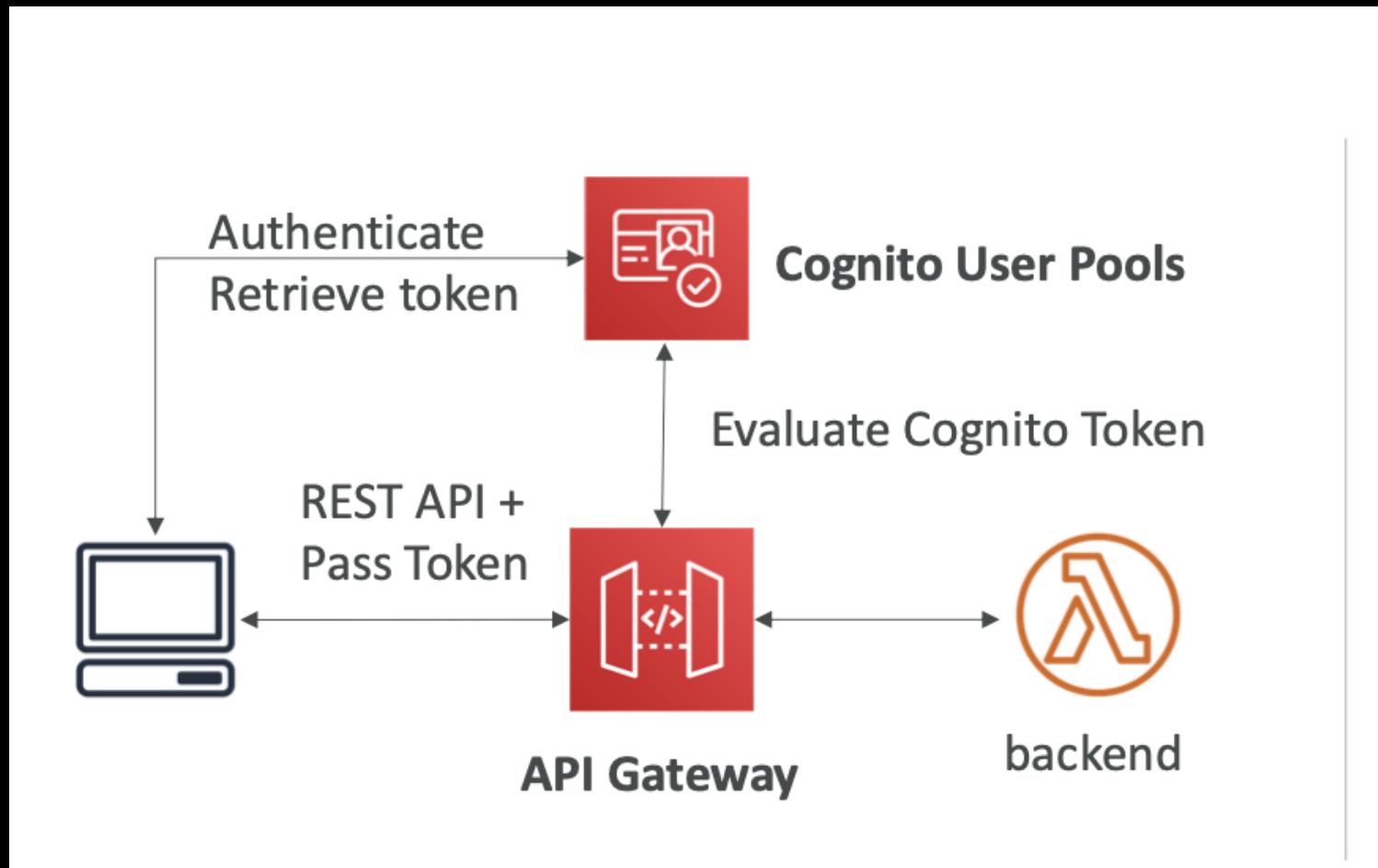
Cognito - Overview

- We want to give users an identity so that they can interact with our applications.
- Cognito User Pools:
 - Sign in functionality for app users.
 - Integrates with API Gateway & Application Load Balancer.
- Cognito Identity Pools (Federated Identity):
 - Provide AWS credentials to users so they can access AWS resources directly.
 - Integrates with Cognito User Pools as an identity provider.
- Cognito vs IAM: “hundreds of users”, “mobile users”.

Cognito User Pools

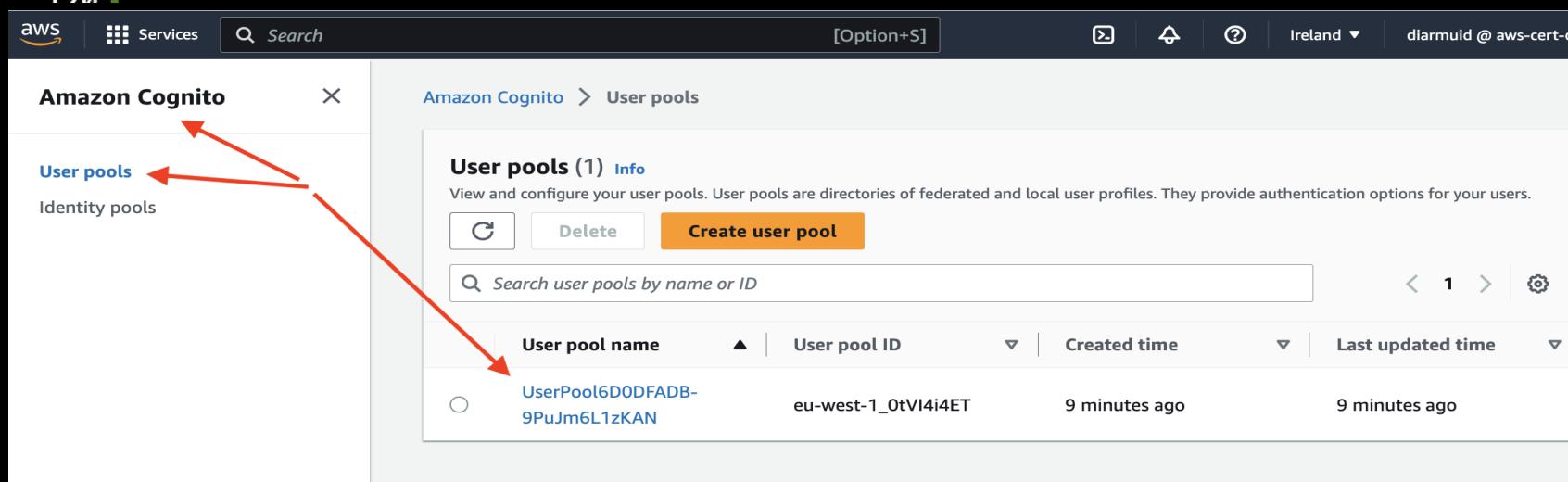
- Creates a serverless 'users database' for your web & mobile apps.
- Built-in features:
 - Simple login: Username (or email) / password combination.
 - Password reset.
 - Email & Phone Number Verification.
 - Multi-factor authentication (MFA).
 - Federated Identities: Facebook, Google, etc
 - Block user account - compromised credentials.
- Include JSON Web Token (JWT) in Login response.

Cognito User Pools

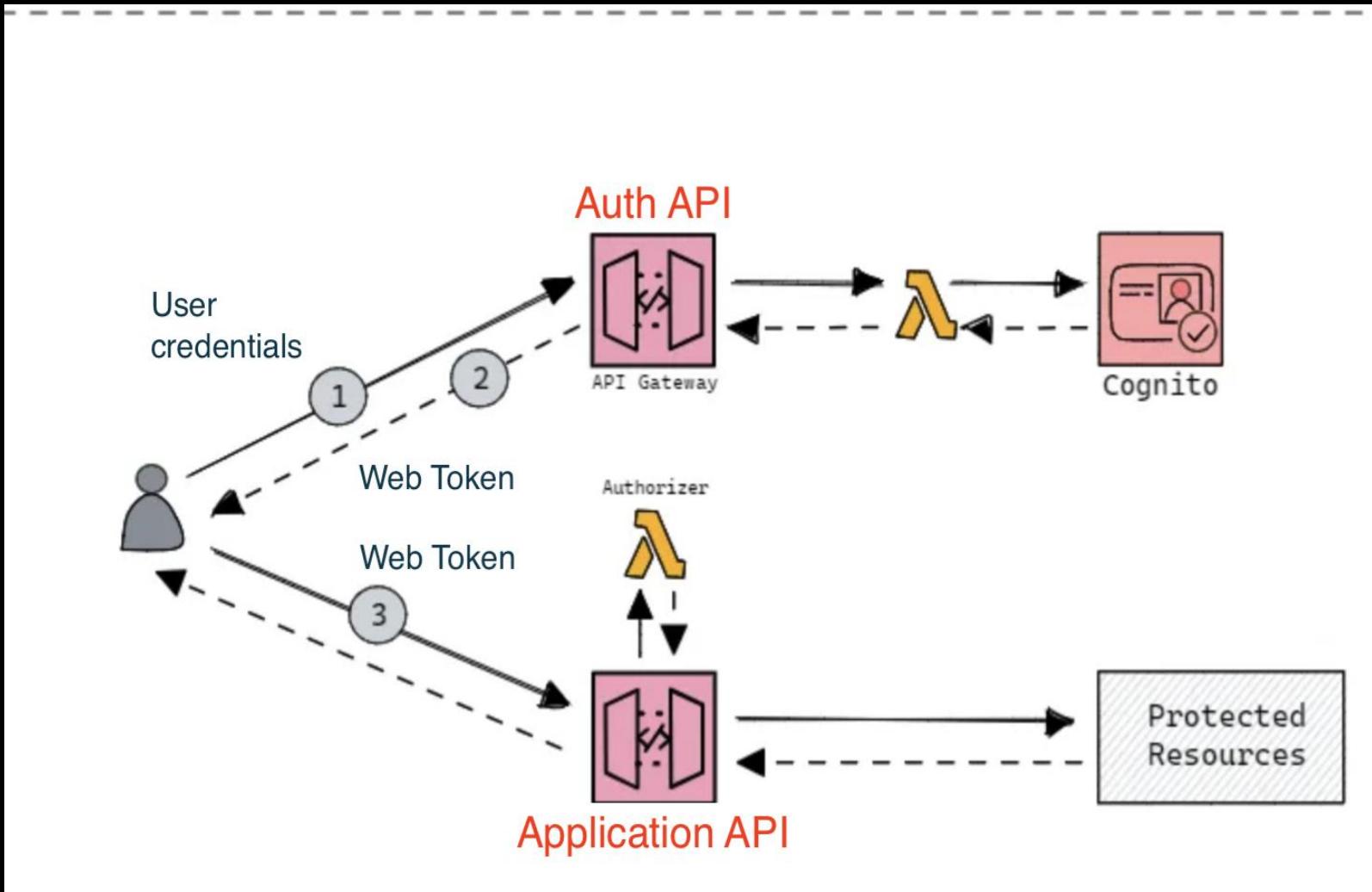


CDK code -

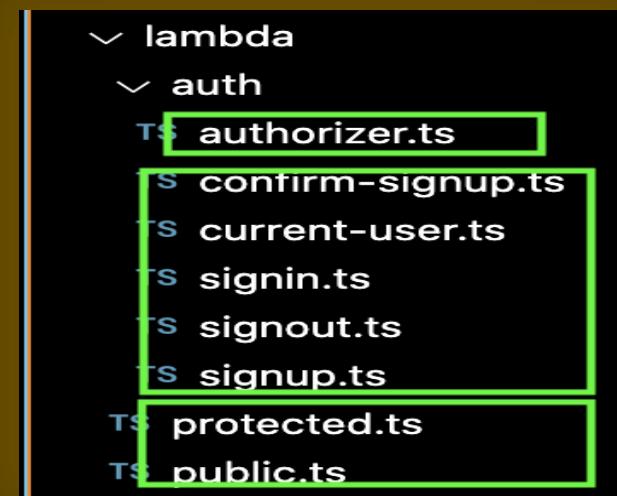
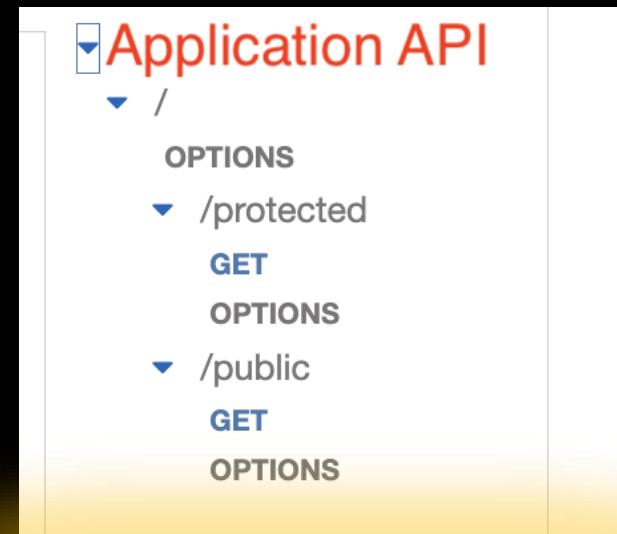
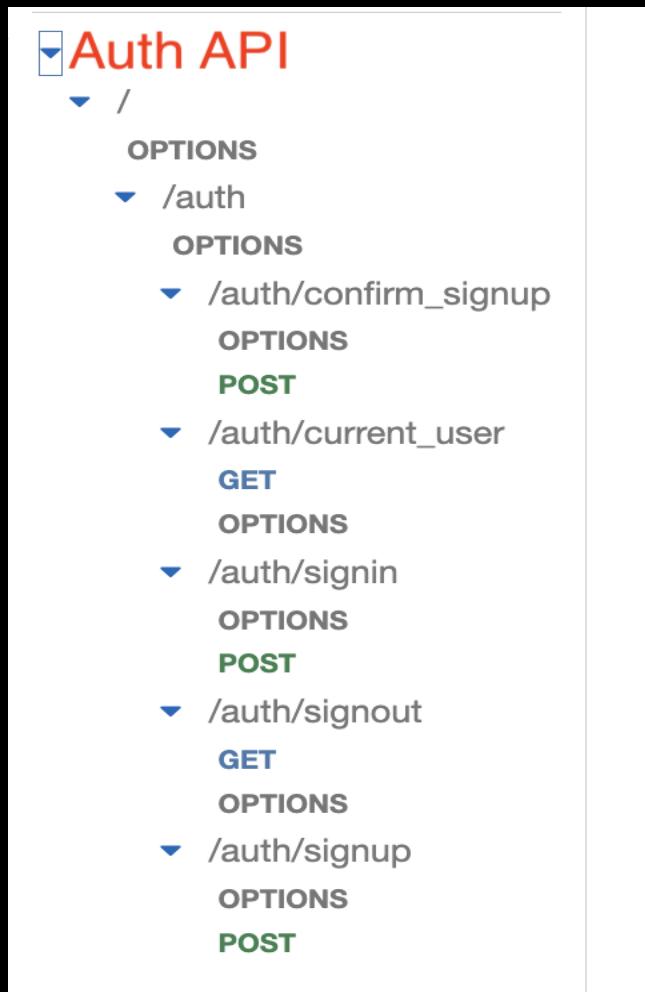
```
109  
110 // CDK setup  
111 const userPool = new UserPool(this, 'UserPool', {  
112   signInAliases: { username: true, email: true },  
113   selfSignUpEnabled: true,  
114   removalPolicy: RemovalPolicy.DESTROY,  
115 });  
116  
117 const appClient = userPool.addClient('AppClient', {  
118   authFlows: { userPassword: true },  
119 });  
120
```



Example – Common Infrastructure setup



Example – The APIs



Example – Signup

The screenshot shows the Postman application interface and a Gmail inbox window.

Postman Interface:

- Header:** Apple Postman, File, Edit, View, Window, Help, Wed 16 Oct 13:44
- Left Sidebar:** Collections (DistributedSystems), Environments, History.
- Middle Panel:** API Network, Cognito collection selected. POST Signup selected. Request method: POST, URL: {{auth_url}} auth/signup. Headers tab shows 8 items. Body tab is active, showing raw JSON:

```
1 "username": "userA",  
2 "password": "passwA!1",  
3 "email": "someone@gmail.com"
```
- Bottom:** Response panel showing a Gmail inbox with an unread message from no-reply@verificationemail.com with subject "Verify your new account". The message contains a verification code: S 105002.

Example – Signup lambda

```
129  
130  const client = new CognitoIdentityProviderClient({ region: "eu-west-1" });  
131  
132  export const handler: APIGatewayProxyHandlerV2 = async (event) => {  
133  
134    const { username, email, password }: eventBody = JSON.parse(event.body);  
135  
136    const params: SignUpCommandInput = {  
137      ClientId: process.env.CLIENT_ID!, ←  
138      Username: username,  
139      Password: password,  
140      UserAttributes: [{ Name: "email", Value: email }],  
141    };  
142  
143    try {  
144      const command = new SignUpCommand(params);  
145      const res = await client.send(command);  
146      return {  
147        statusCode: 200,  
148        body: JSON.stringify({  
149          message: res,  
150        }),  
151      };  
152    } catch (err) {....}  
153  };
```

Example – Confirm Signup

Postman File Edit View Window Help Wed 16 Oct 13:50

Home Workspaces API Network

DistributedSystems New Import

Cognito POST Signup POST Confirm POST SignIn GET Protected GET Public

POST {{auth_url}} /auth/confirm_signup

Params Auth Headers (8) Body Scripts Settings

raw Text

```
1 {  
2   "username": "userA",  
3   "code": "256349"  
4 }
```

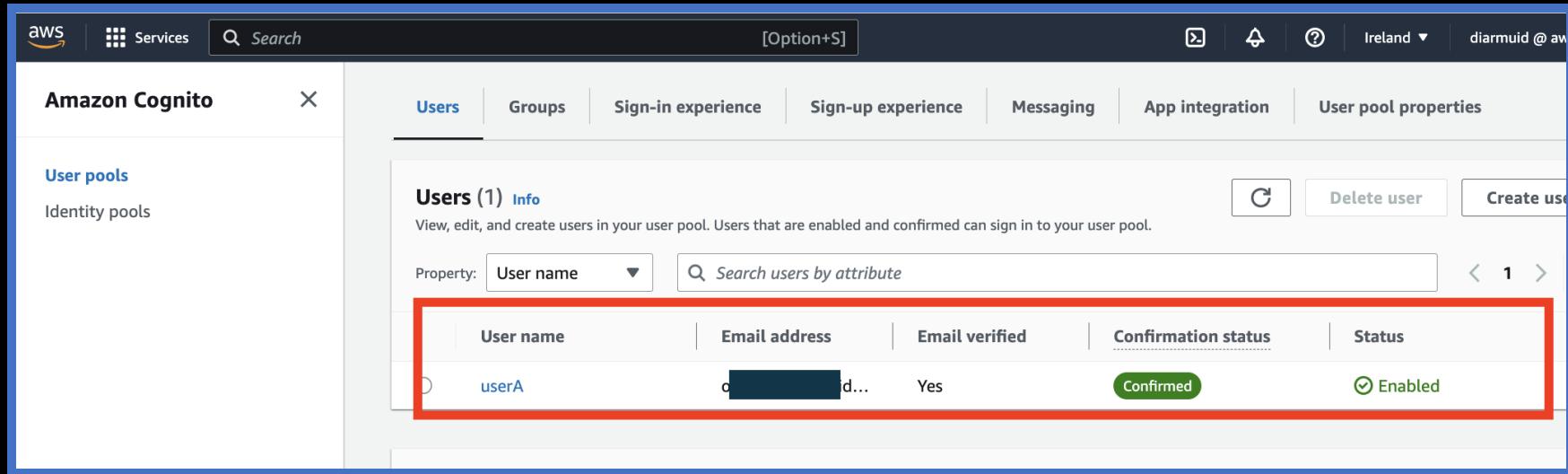
Save Share Send

Response

Click Send to get a response

Online Find and replace Console Postbot Runner Start Proxy Cookies Vault Trash ?

Example – Confirm Signup (Cognito User pool)



The screenshot shows the AWS Cognito User Pools console. The left sidebar has 'User pools' selected. The main area shows a table of users with one entry:

| User name | Email address | Email verified | Confirmation status | Status |
|-----------|---------------|----------------|---------------------|---------|
| userA | [REDACTED] | Yes | Confirmed | Enabled |

A red box highlights the entire row for 'userA'. The 'Confirmation status' column shows 'Confirmed' and the 'Status' column shows 'Enabled'.

Example – Confirm Signup lambda

```
157 const client = new CognitoIdentityProviderClient({ region: "eu-west-1" });
158
159 type eventBody = { username: string; code: string };
160
161 export const handler: APIGatewayProxyHandlerV2 = async (event) => {
162
163   const { username, code }: eventBody = JSON.parse(event.body);
164
165   const params: ConfirmSignUpCommandInput = {
166     ClientId: process.env.CLIENT_ID!,
167     Username: username,
168     ConfirmationCode: code,
169   };
170
171   try {
172     const command = new ConfirmSignUpCommand(params);
173     const res = await client.send(command);
174
175     return {
176       statusCode: 200,
177       body: JSON.stringify({
178         message: `User ${username} successfully confirmed`,
179         confirmed: true,
180       }),
181     };
182   } catch (err) { ... }
183 }
```

Example – Sign in lambda

```
186
187 const client = new CognitoIdentityProviderClient({ region: "eu-west-1" });
188
189 export const handler: APIGatewayProxyHandlerV2 = async (event) => {
190   const { username, password } = JSON.parse(event.body);
191   const params: InitiateAuthCommandInput = {
192     ClientId: process.env.CLIENT_ID!,
193     AuthFlow: "USER_PASSWORD_AUTH",
194     AuthParameters: {
195       USERNAME: username,
196       PASSWORD: password,
197     },
198   };
199   try {
200     const command = new InitiateAuthCommand(params);
201     const { AuthenticationResult } = await client.send(command);
202     const token = AuthenticationResult.IdToken;
203
204     return {
205       statusCode: 200,
206       headers: {
207         "Access-Control-Allow-Headers": "*",
208         "Access-Control-Allow-Origin": "*",
209         "Set-Cookie": `token=${token}; SameSite=None; Secure; HttpOnly; Path=/; M
210       },
211       body: JSON.stringify({
212         message: "Auth successfull",
213         token: token,
214       }),
215     };
216   } catch (err) {.... }
217 };
```

ⓘ Restart Visual Studio Code to apply the la

Update Now

Example – Sign in request/response

The screenshot shows the Postman application interface. On the left, the sidebar displays collections like 'DistributedSystems', environments, and history. The main workspace shows a POST request to 'Cognito / SignIn'. The request method is 'POST' and the URL is '{{auth_url}} /auth/signin'. The 'Body' tab is selected, showing a JSON payload:

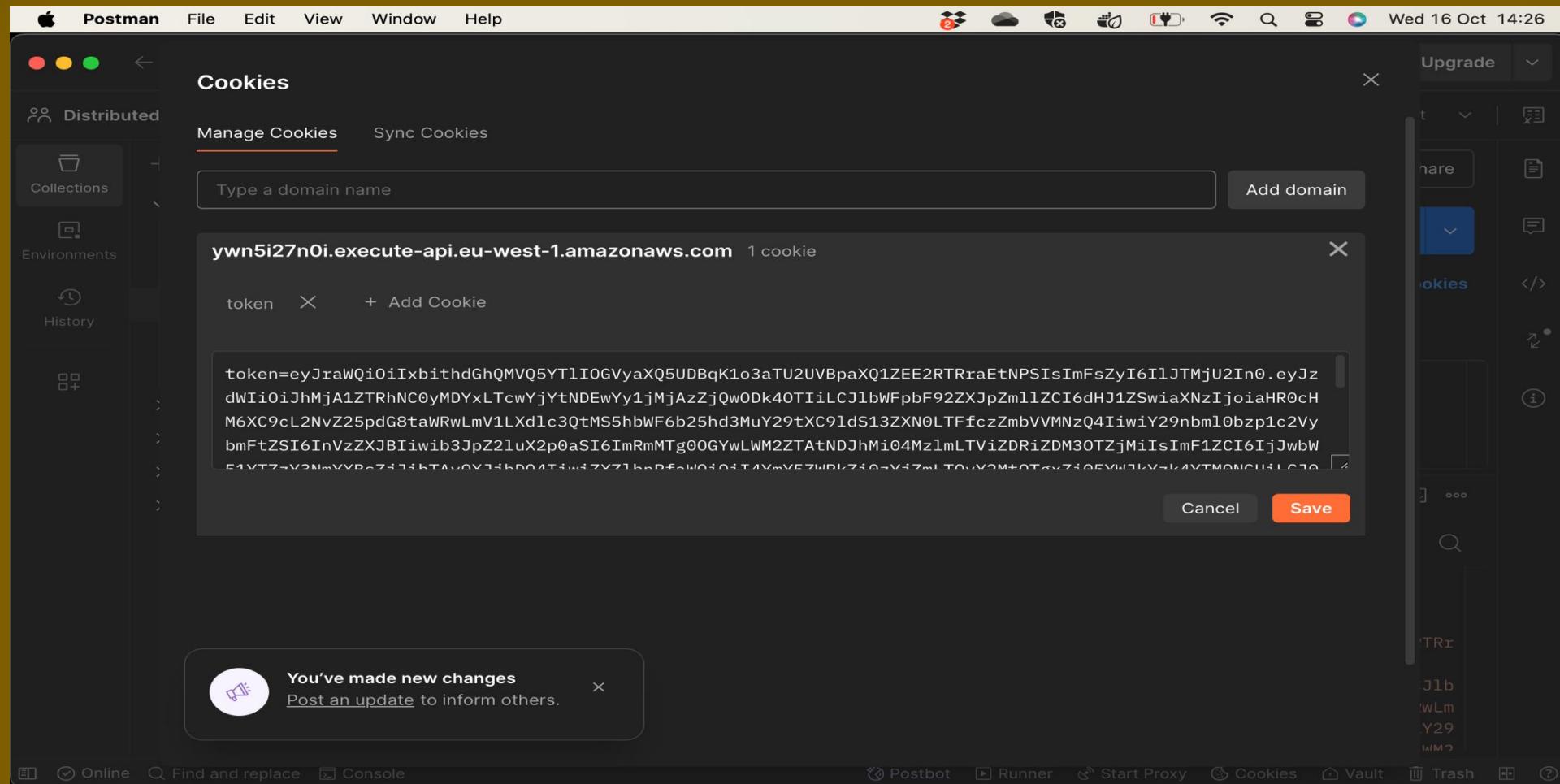
```
1 {
2   "username": "userA",
3   "password": "passWA!1"
4 }
```

The response status is '200 OK' with a response time of '1901 ms' and a size of '2.56 KB'. The response body is displayed in 'Pretty' format:

```
1 {
2   "message": "Auth successful",
3   "token": "eyJraWQiOiIxbithdGhQMVQ5YTlI0GVyaXQ5UDBqK1o3aTU2UVBpaXQ1ZEE2RTRxaEtNPSIsImFsZyI6I1JTMjU2In0.eyJzdWIiOiJhMjA1ZTRhNC0yMDYxLTcwYjYtNDEwYy1jMjAzZjQwODk4OTIiLCJ1bWFpbF92ZXJpZmlzCI6dHJ1ZSwiaXNzIjoiaHR0cHM6XC9cL2NvZ25pdG8taWRwLmV1LXd1c3QtMS5hbWF6b25hd3MuY29tXC91dS13ZXN0LTFfczZmbVVVMNzQ4IiwiY29nbmVhZG1sc2VubmE+7cTATpV7V7DT4w4h2n721uv2n0nST4TmPmMTg0OCVwIWM2
```

A red arrow points from the 'token' field in the response body back to the 'Cookies' tab in the request settings, indicating that the token is being stored as a cookie.

Example – Postman Cookie Management



Example – Decoded JWT token

The screenshot shows a Chrome browser window displaying the jwt.io website. The URL in the address bar is `jwt.io`. The page has a dark theme with a pink header bar.

Encoded: PASTE A TOKEN HERE

```
eyJraWQiOiJiejFcL21HV05LVWcxerUNaZVd5eGR  
tdkVZSWhRdUlpu1z1THNyU01YQkF2VT0iLCJhbG  
ci0iJSUzI1NiJ9.eyJzdWIiOiIzNT1lMjE1Yi0y  
ZTA1LTRiN2YtOTEzNS1hMTAxYjhhNDg2MzEiLCJ  
1bWFpbF92ZXJpZmllZCI6dHJ1ZSwiaXNzIjoiaH  
R0cHM6XC9cL2NvZ25pdG8taWRwLmV1LXd1c3QtM  
S5hbWF6b25hd3MuY29tXC91dS13ZXN0LTFFfcHRH  
WE9IM1VyIwiY29nbml0bzp1c2VybmrZSI6InV  
zZXJBIiwib3JpZ2luX2p0aSI6IjIwM2UwMDNhLT  
E4YWItNGRjOS1iODU3LTfKOGNiNjMzNWZkNiIsI  
mF1ZCI6IjM2M2NycGFoMmdhbDRmY2JsbDFyZGN1  
YmIxIiwiZXlbnRfaWQiOiI4MWY5NmY3ZS1hMmF  
kLTQ2N2EtYmVjMy0zODY2M2N10TlmZTMiLCJ0b2  
t1b191c2Ui0iJpZCIsImF1dGhfdGltZSI6MTY4N  
zI2NTg2NiwiZXhwIjoxNjg3MjY5NDY2LCJpYXQi  
0jE20DcyNjU4NjYsImp0aSI6IjY3N2Y10WU5Ltc  
xNDUtNDczMy05NjI3LTk1MmVhYTliZDFmNSIsIm
```

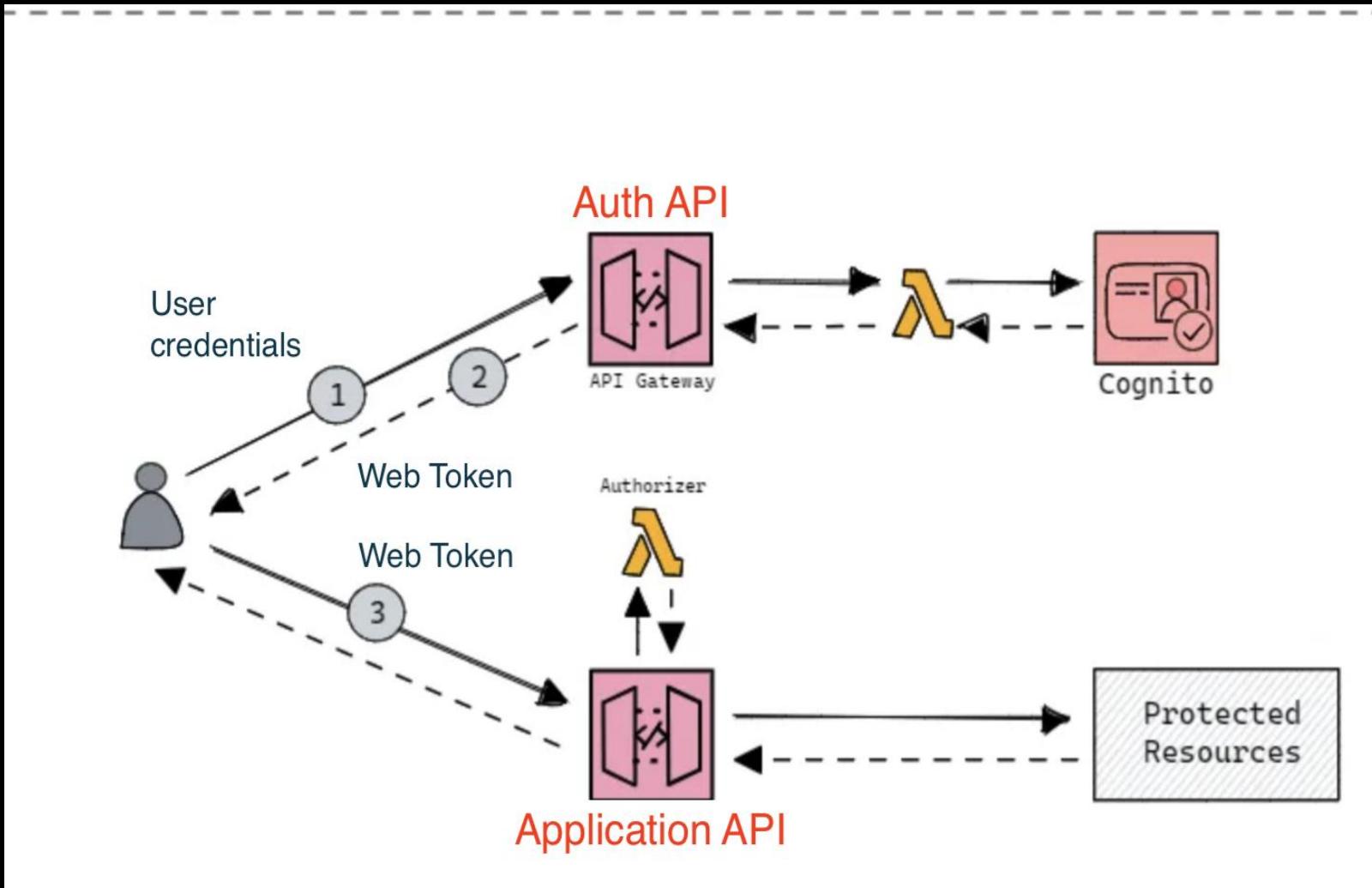
Decoded: EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
"alg": "RS256"  
}
```

PAYOUT: DATA{
"sub": "359e215b-2e05-4b7f-9135-a101b8a48631",
"email_verified": true,
"iss": "https://cognito-idp.eu-west-
1.amazonaws.com/eu-west-1.ptGXOH3Ur",
"cognito:username": "userA",
"origin_jti": "203e003a-18ab-4dc9-b857-1d8cb6335fd6",
"aud": "363crpah2gal4fcbl1rdcubb1",
"event_id": "81f9ef7e-a2ad-467a-bec3-38663ce99fe3",
"token_use": "id",
"auth_time": 1687265866,
"exp": 1687269466,
"iat": 1687265866,
"jti": "677f59e9-7145-4733-9627-952eaa9bd1f5",
"email": "oconnordarmuid@gmail.com"

Example – Common Infrastructure setup



Example – App API IaC (CDK)

```
230 const api = new RestApi(this, "App Api", ....);
231
232 const protectedRes = api.root.addResource("protected");
233 const publicRes = api.root.addResource("public");
234
235 const protectedFn = new NodejsFunction(this, "ProtectedFn", ....);
236 const publicFn = new NodejsFunction(this, "PublicFn", ....);
237
238 const authorizerFn = new NodejsFunction(this, "AuthorizerFn", ....);
239
240 const requestAuthorizer = new RequestAuthorizer(this, "RequestAuthorizer", {
241   identitySources: [IdentitySource.header("cookie")],
242   handler: authorizerFn,
243   resultsCacheTtl: Duration.minutes(0),
244 });
245
246 protectedRes.addMethod("GET", new LambdaIntegration(protectedFn), {
247   authorizer: requestAuthorizer,
248   authorizationType: AuthorizationType.CUSTOM,
249 });
250
251 publicRes.addMethod("GET", new LambdaIntegration(publicFn));
```



Example – App API Protected endpoint request

The screenshot shows the Postman application interface with a dark theme. On the left, the sidebar displays collections like 'DistributedSystems', 'Cognito' (which is expanded), and others. Under 'Cognito', there are several requests: 'POST Signup', 'POST Confirm', 'POST SignIn', 'GET Protected' (which is selected), and 'GET Public'. The main workspace shows a 'Protected' endpoint for 'Cognito' using a GET method. The URL is {{app_url}}/protected. The 'Headers' tab is selected, showing the following configuration:

- Host: `POSTMAN-TOKEN` (with a checked checkbox)
- User-Agent: `PostmanRuntime/7.42.0` (unchecked)
- Accept: `/*/*` (unchecked)
- Accept-Encoding: `gzip, deflate, br` (unchecked)
- Connection: `keep-alive` (unchecked)
- Cookie: `{{token}}` (with a checked checkbox)

Red arrows point from the text labels 'Host' and 'Cookie' to their corresponding entries in the Headers table. At the bottom of the interface, a message box says 'You've made new changes' and 'Post an update to inform others.' A status bar at the bottom right indicates a successful response: '200 OK'.

Example – App API Request Authorizer.

- Steps performed by authorizer:
 1. Parse HTTP Request Cookie header value & store entries in a local Map data structure.
 2. Find the ‘token’ key in the Map.
 3. If not found:
 - Return an IAM policy Denying use of the App Web API.
 4. Verify the token - Decode and check user exists in the User pool.
 5. If successful verification:
 - Return IAM policy that Allows execution of the App Web API.
 - Else
 - Return policy Denying execution of the App Web API.