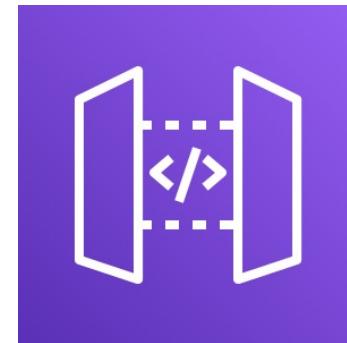




# Serverless Web APIs (Continued)



# API Gateway

# Components of a Serverless app

Logic



Lambda

State



RDS



DynamoDB



Amazon S3

Communication



API Gateway



Kinesis



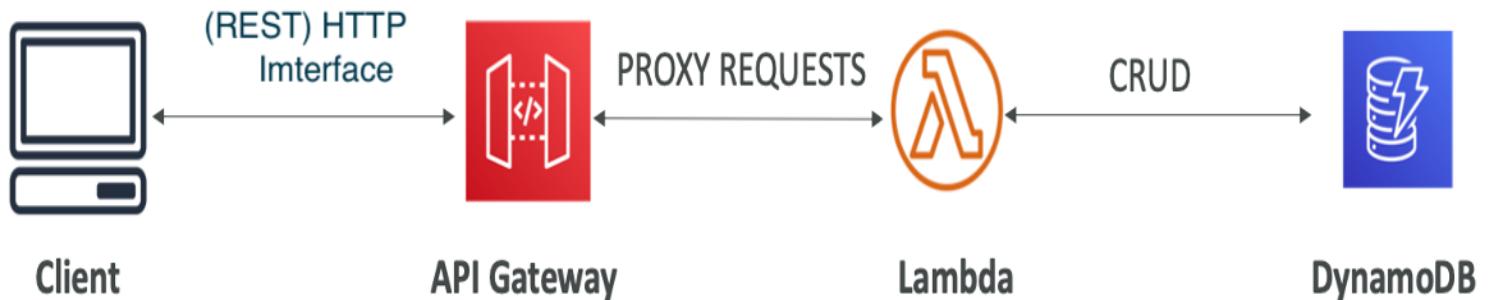
SNS



SQS

# API Gateway

## Three-tier serverless Web API

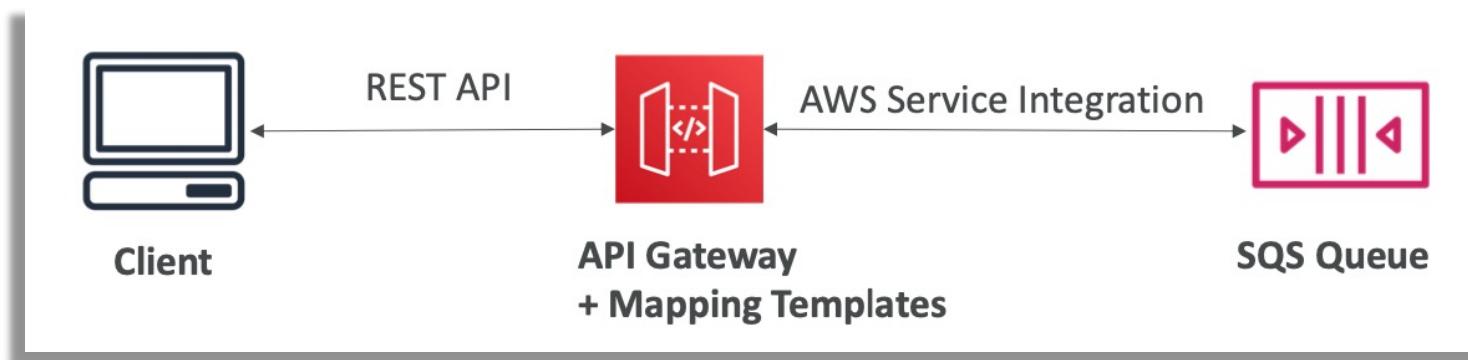


# API Gateway

- A fully managed (aka Serverless) service for Web APIs.
- Creating REST API endpoints.
- Support for the WebSocket Protocol.
- Handle API versioning (v1, v2...).
- Handle different environments (dev, test, prod).
- Handle security (Authentication and Authorization).
- Create API keys; handles request throttling.
- Swagger / Open API import to quickly define APIs.
- Transform and validate requests and responses.
- Generate SDK and API specifications.
- Cache API responses.
- Supports DDos protection.

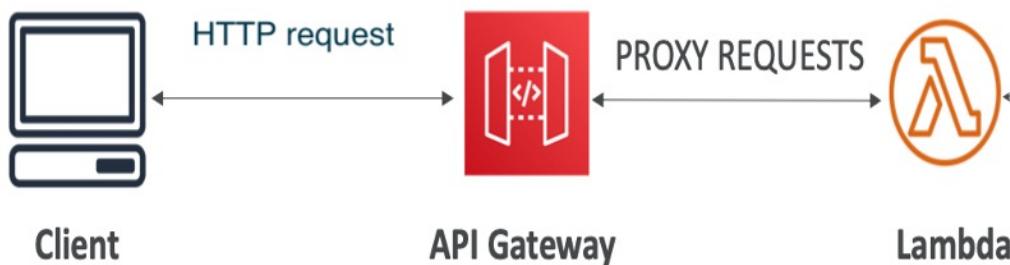
# API Gateway - Integration Types

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



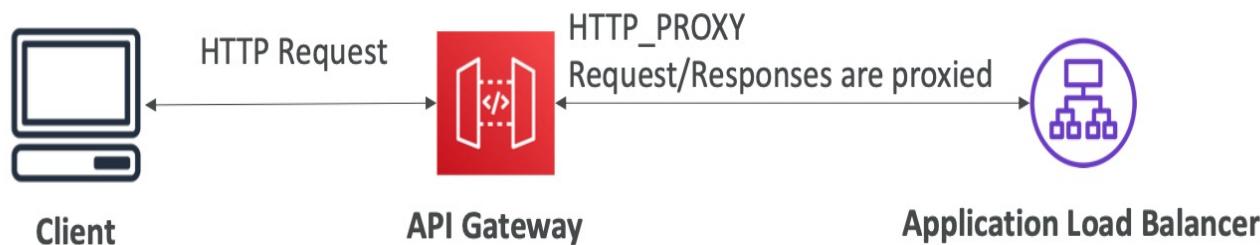
# API Gateway - Integration Types

- **Integration Type AWS\_PROXY (Lambda Proxy):**
  - incoming request from the client is the input to Lambda.
  - No mapping template.
  - HTTP headers, query string parameters are passed as arguments.



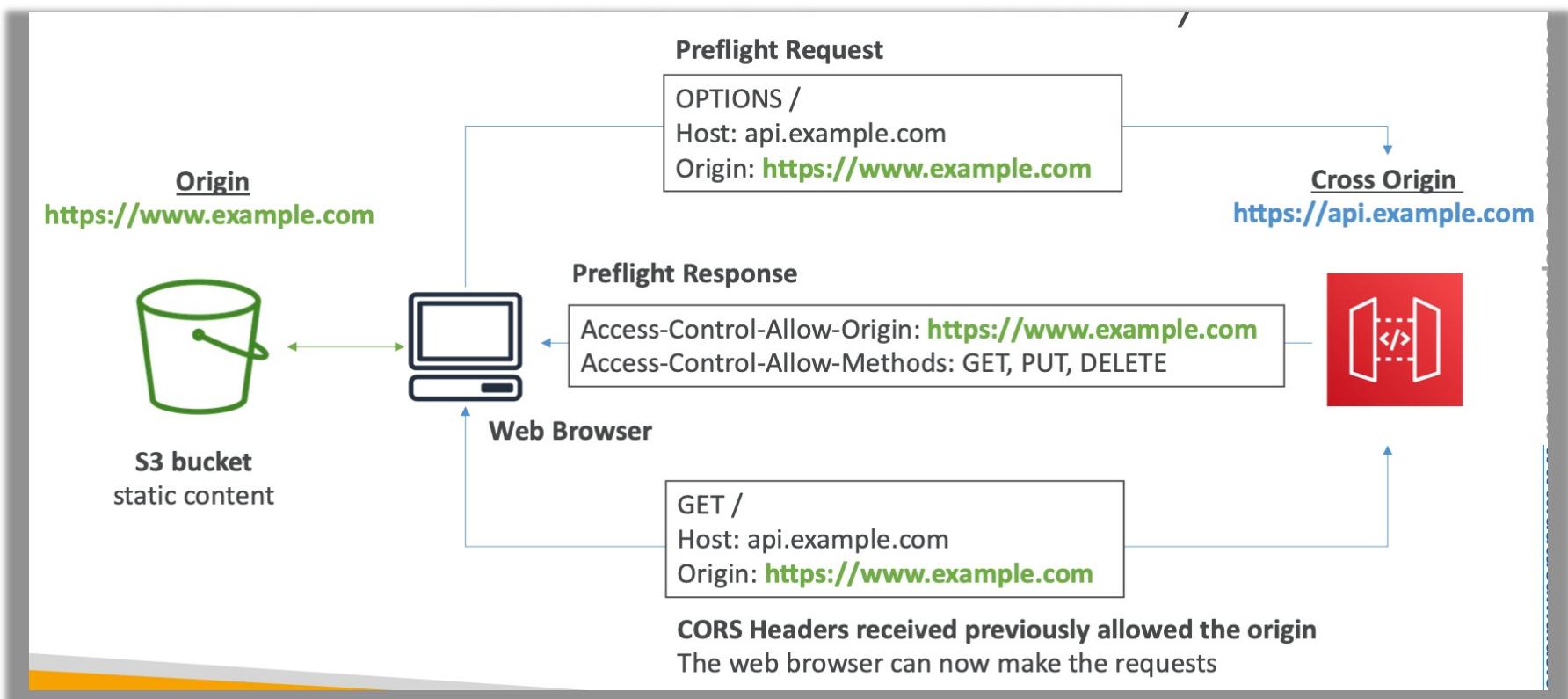
# API Gateway - Integration Types

- **Integration Type HTTP\_PROXY**
  - 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



# API Gateway - CORS

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



# API Gateway -Example

- **Objective – Web API for managing TODOs.**
- **Two Endpoints**
  - **/todos**
    - **GET – Read all todos**
    - **POST – Add new TODO**
  - **/todos/{todoID}**
    - **GET – Read specific TODO**

# API Gateway - Example

The screenshot illustrates the AWS API Gateway interface through two distinct views:

**Top View: APIs Overview**

- Left Sidebar:** Contains links for "APIs", "Custom domain names", and "VPC links".
- Center Content:** A table titled "APIs (1)" lists the single entry "DemoAPI".

Name	Description	ID	Protocol
DemoAPI	example api gateway	ganf19s9t8	REST
- Header:** Includes the AWS logo, "Services" menu, a search bar, and a "[Option+S]" button.

**Bottom View: Method Execution Details**

- Left Sidebar:** Shows "APIs", "Custom Domain Names", "VPC Links", and the selected "API: DemoAPI".
- Center Content:** Displays the "Resources" section for the "/todos" endpoint, specifically the "POST" method.
  - Resource Tree:** Shows the hierarchy: / todos /{todoId} POST.
  - Actions:** Includes a "TEST" button and a "Method Request" panel.
  - Method Request Panel:** Shows "Auth: NONE" and "ARN: arn:aws:execute-api:eu-west-1:517039770760:ganf19s9t8/\*P".
  - Integration Request Panel:** Shows "Type: LAMBDA\_PROXY".
- Header:** Includes the AWS logo, "Services" menu, a search bar, and a "[Option+S]" button. The top navigation bar also shows the path: "APIs > DemoAPI (ganf19s9t8) > Resources > /todos (ycf4jb) > POST".

```

// 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
);

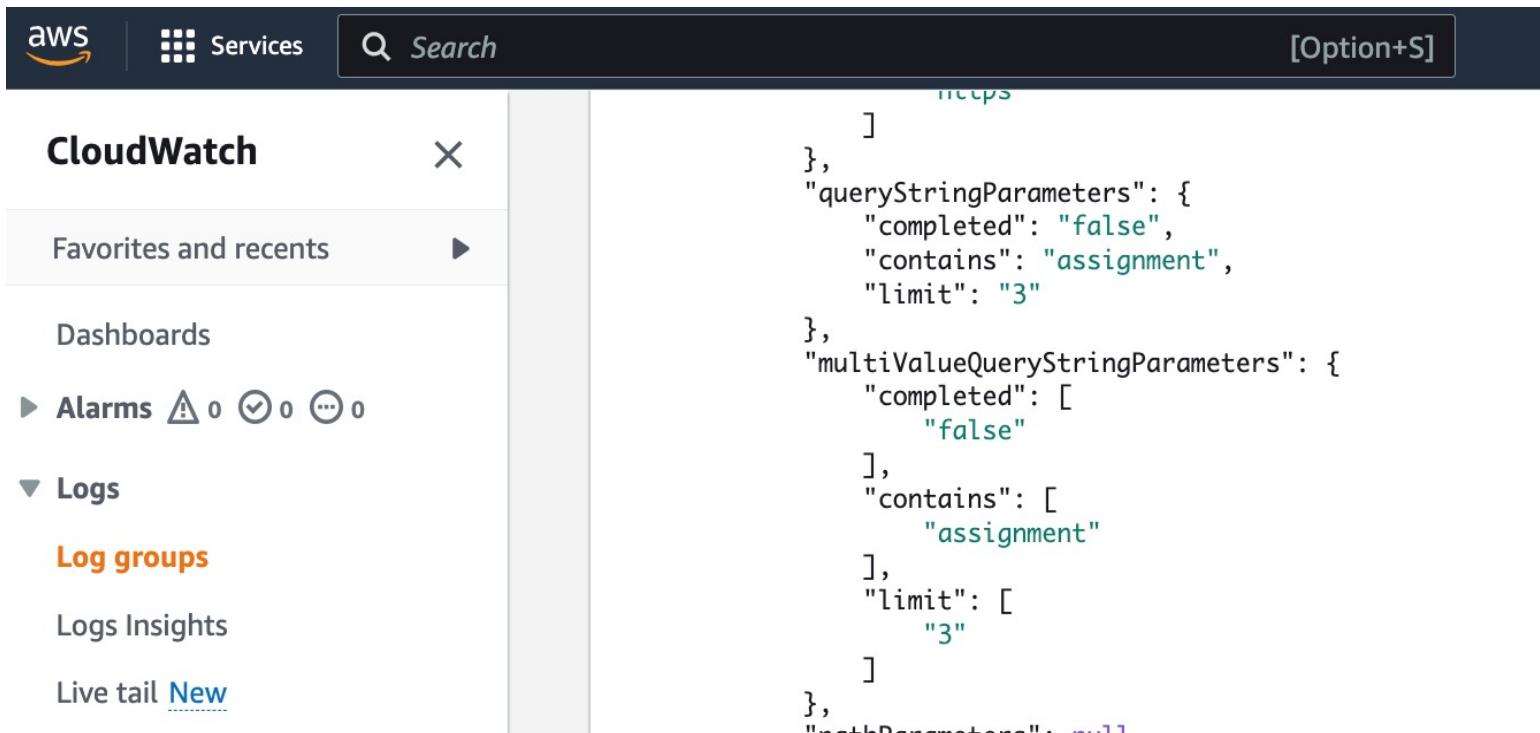
const todoDetailsEndpoint = todosEndpoint.addResource("{todoId}");
todoEndpoint.addMethod(
  "GET",
  new LambdaIntegration(RESTEndpointFn, { proxy: true })
);

```

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

# API Gateway - Example

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

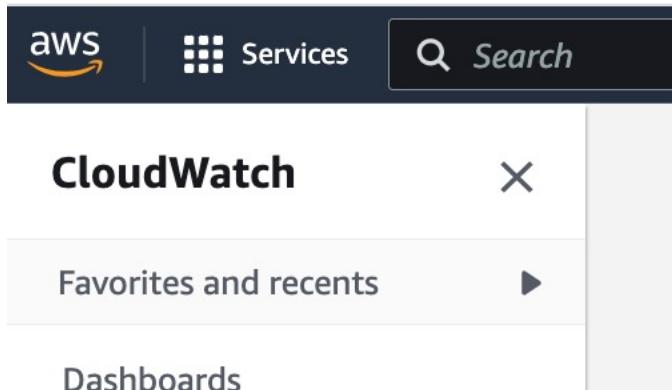


The screenshot shows the AWS CloudWatch Metrics interface. The top navigation bar includes the AWS logo, 'Services' button, a search bar with placeholder 'Search', and a keyboard shortcut '[Option+S]'. The left sidebar has a 'CloudWatch' title and a 'Logs' section expanded, showing 'Log groups', 'Logs Insights', and 'Live tail' options. The main content area displays a single log entry with the following JSON payload:

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

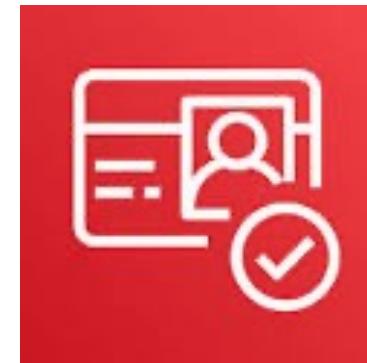
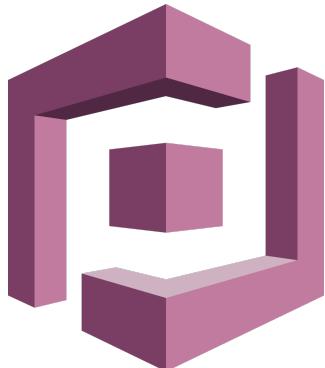
# API Gateway - Example

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



```
CloudWatch
Favorites and recents
Dashboards
```

```
        },
        "queryStringParameters": null,
        "multiValueQueryStringParameters": null,
        "pathParameters": {
            "todoId": "1234"
        },
        "stageVariables": null,
        "requestContext": {
```



# Amazon Cognito

# Components of a Serverless app

## Logic



Lambda

## State



RDS



DynamoDB



Amazon S3



Cognito

## Communication



API Gateway



Kinesis



SNS



SQS

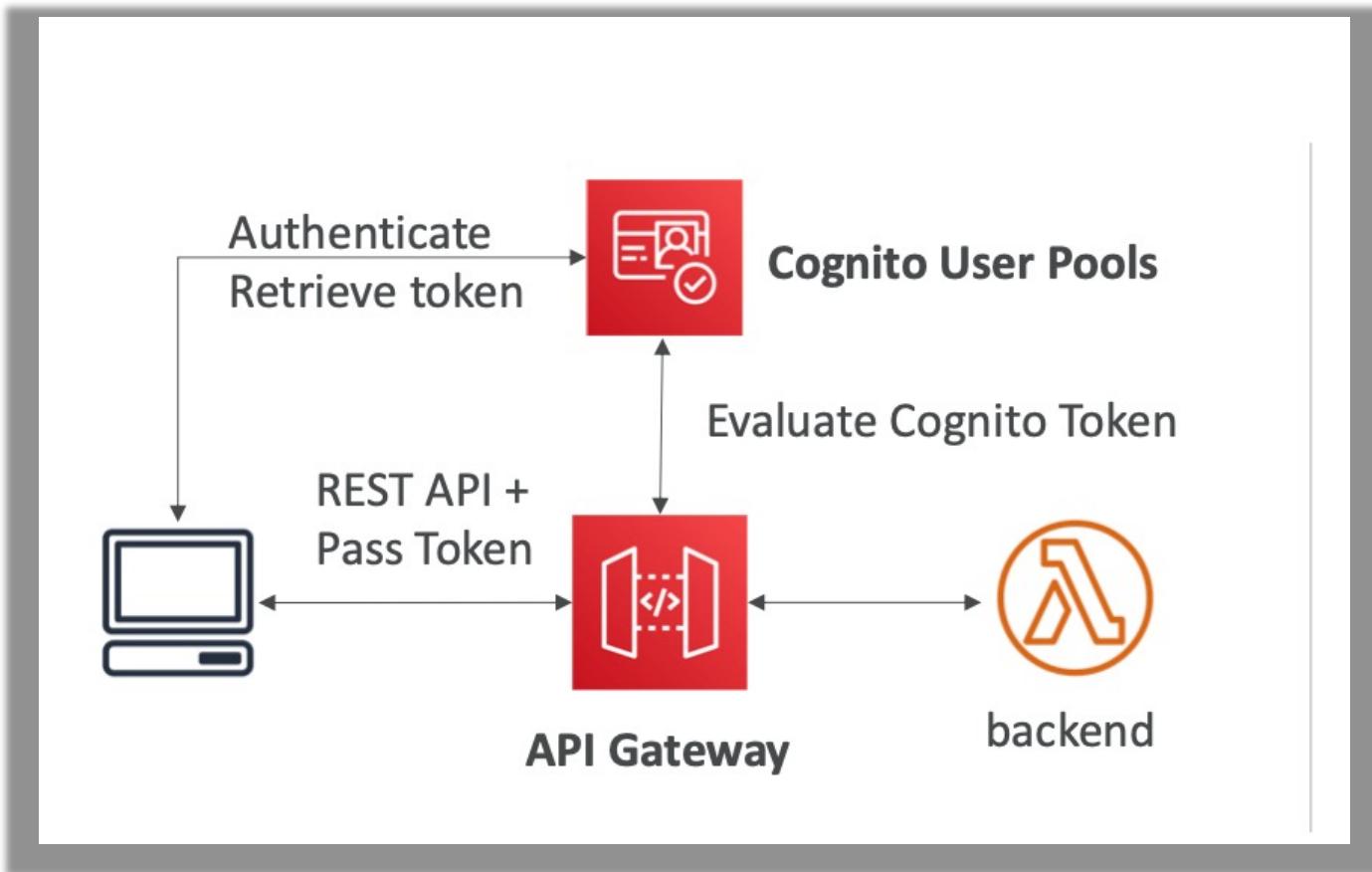
# Amazon Cognito

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

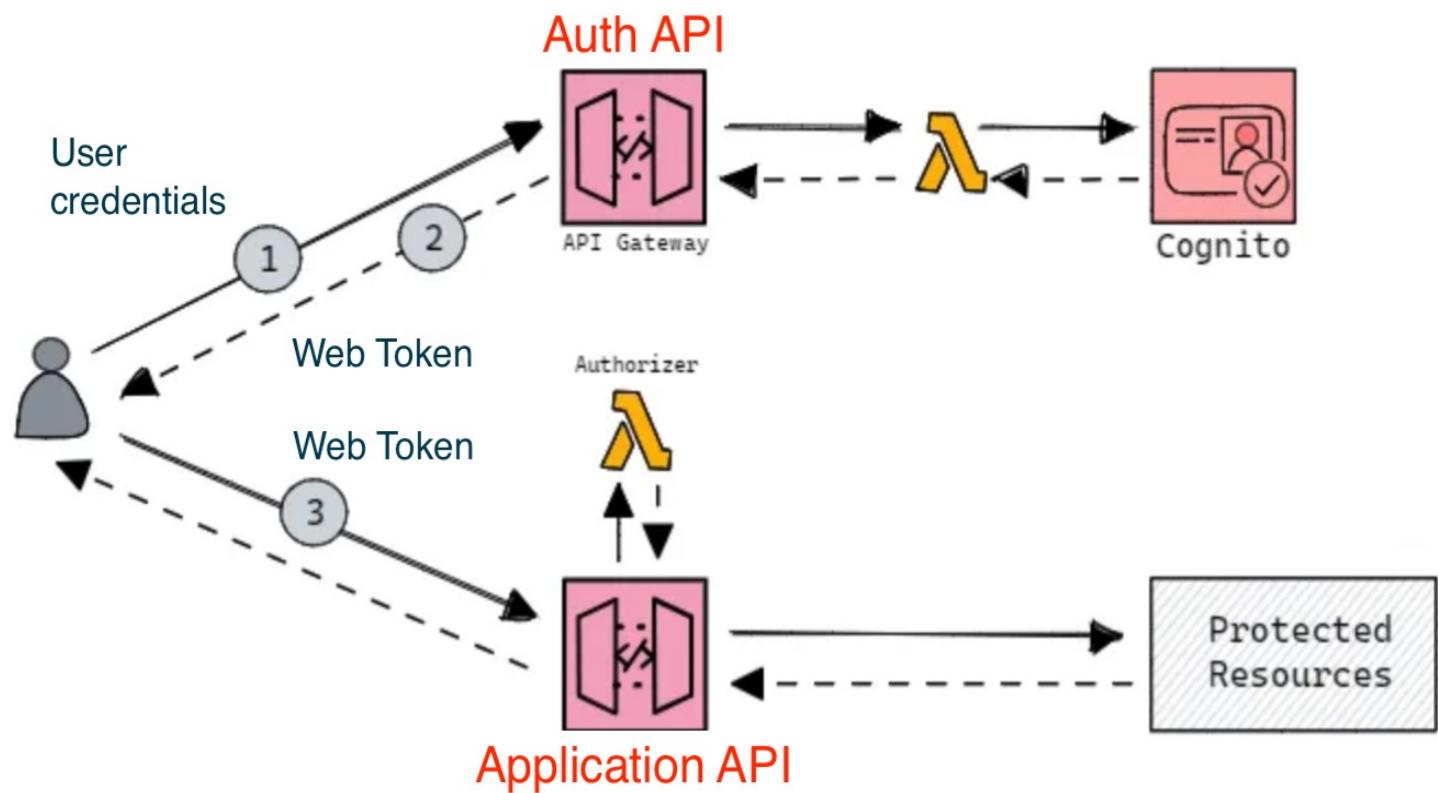
# Cognito User Pool

- Creates a serverless database of user for your web & mobile apps.
- Simple login: Username (or email) / password combination.
- Password reset.
- Email & Phone Number Verification.
- Multi-factor authentication (MFA).
- Federated Identities: Facebook, Google...
- Feature: block users if their credentials are compromised elsewhere
- Include JSON Web Token (JWT) in Login response.

# Cognito User Pool



# Example



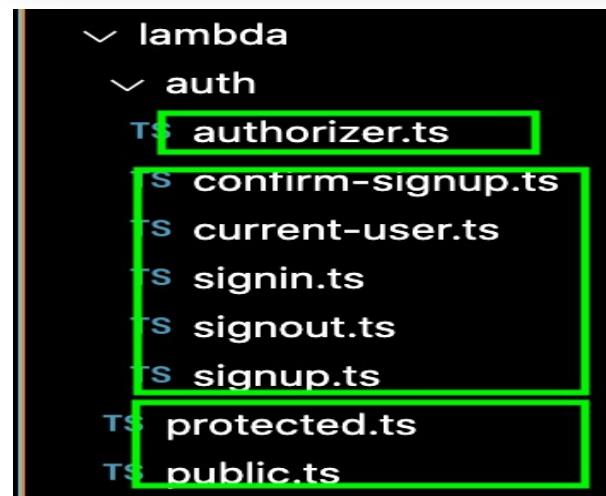
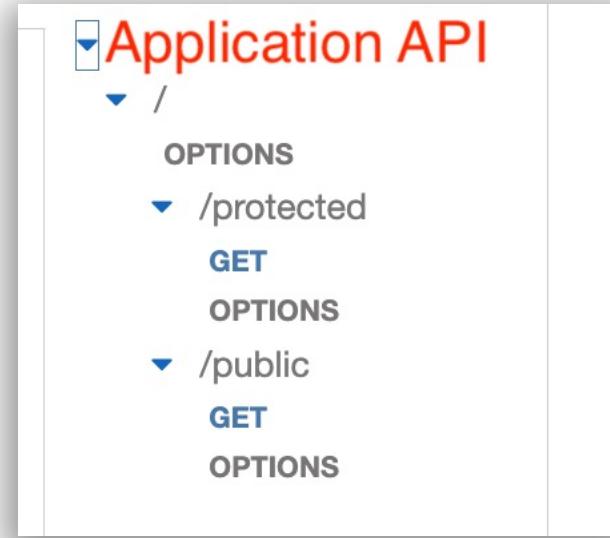
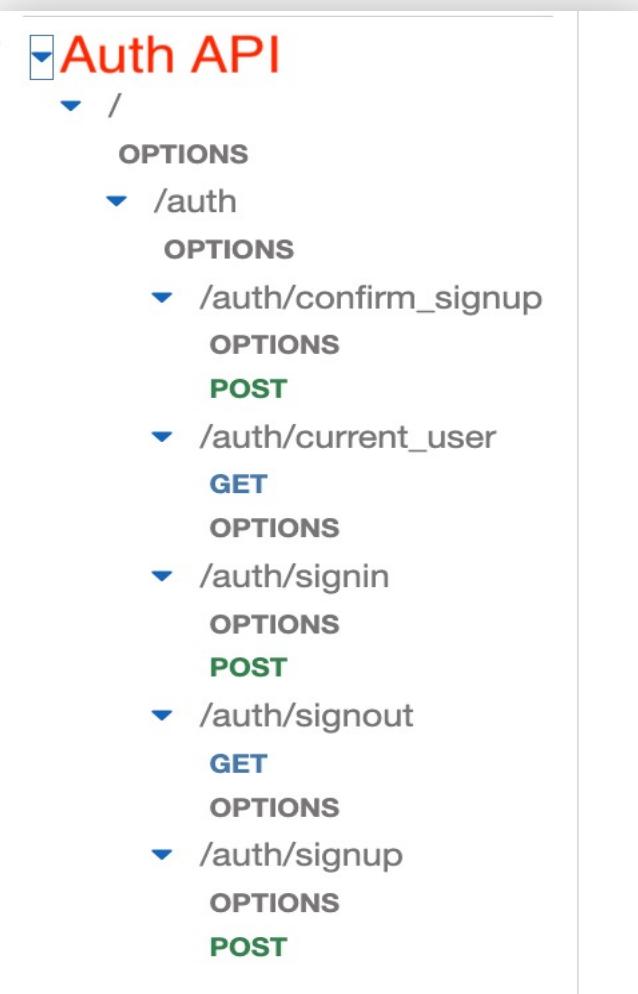
# Example – User Pool

```
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
```

The screenshot shows the AWS Cognito User Pools console. On the left, there's a sidebar with 'User pools' and 'Identity pools'. The main area shows a list of user pools with one item: 'User pool name: UserPool6D0DFADB-9PuJm6L1zKAN', 'User pool ID: eu-west-1\_0tVI4i4ET', 'Created time: 9 minutes ago', and 'Last updated time: 9 minutes ago'. A red arrow points from the 'User pools' link in the sidebar to the list item in the main area.

User pool name	User pool ID	Created time	Last updated time
UserPool6D0DFADB-9PuJm6L1zKAN	eu-west-1_0tVI4i4ET	9 minutes ago	9 minutes ago

# Example – APIs



# Example – SignUp

The screenshot shows a POST request to <https://87jqug6skd.execute-api.eu-west-1.amazonaws.com/prod/auth/signUp>. The Body tab is selected, showing a JSON payload:

```
1 {  
2   "username": "userA",  
3   "password": "passABCDE!2",  
4   "email": "o[REDACTED]@gmail.com"  
5 }
```

An orange arrow points to the password field. Below the request, the response body is displayed:

```
1 {  
2   "message": {  
3     "$metadata": {  
4       "httpStatusCode": 200,  
5       "requestId": "ad8071ef-",  
6       "attempts": 1,  
7     }  
8   }  
9 }
```

A modal window from the Gmail inbox is overlaid on the interface, showing a verification email from no-reply@verificationemail.com. The subject is "Verify your new account". The message body contains a circled verification code: **s 105002**.

# Example – SignUp

```
125
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

The screenshot shows the Postman application interface for a POST request to a AWS Lambda endpoint.

**Request Details:**

- Method:** POST
- URL:** `https://87jqug6skd.execute-api.eu-west-1.amazonaws.com/prod/auth/confirm_signup`
- Body:** `Text` (selected)
- Body Content:**

```
1 {  
2   "username": "userA",  
3   "code": "105002"  
4 }
```

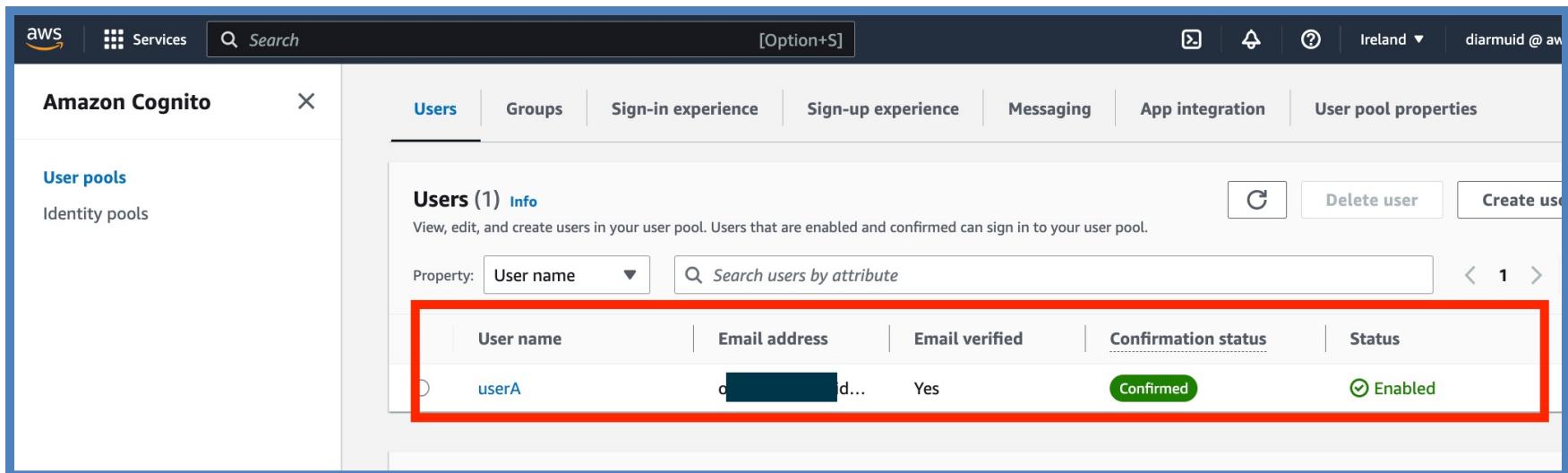
A large orange arrow points from the text "105002" in the Body content to the "code" field in the JSON response below.

**Response Details:**

- Status:** 200 OK
- Time:** 864 ms
- Size:** 374 B
- Save Response:** Available
- Content Type:** JSON (selected)
- Content:**

```
1 {  
2   "message": "User userA successfully confirmed",  
3   "confirmed": true  
4 }
```

# Example – Confirm SignUp



The screenshot shows the AWS Cognito console interface. The top navigation bar includes the AWS logo, Services dropdown, search bar, and user information (Ireland, diarmuid @ aw). Below the navigation is a header with tabs: Users (selected), Groups, Sign-in experience, Sign-up experience, Messaging, App integration, and User pool properties. On the left, a sidebar titled 'User pools' shows '1 User pool' and 'Identity pools'. The main content area is titled 'Users (1) Info' and contains a table with one row. The table columns are: User name, Email address, Email verified, Confirmation status, and Status. The row for 'userA' has the following values: userA, [redacted]@d..., Yes, Confirmed, and Enabled. A red box highlights the entire table row for 'userA'.

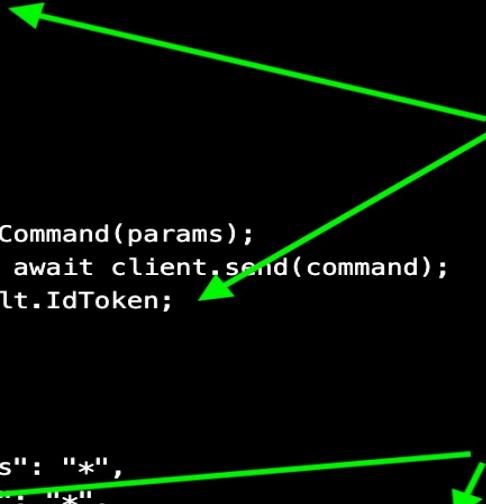
User name	Email address	Email verified	Confirmation status	Status
userA	[redacted]@d...	Yes	Confirmed	Enabled

# Example – Confirm SignUp

```
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) [<...>] You, 1 second ago • Uncommitted changes
183 };
184
```

# Example – SignIn

```
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 }
```

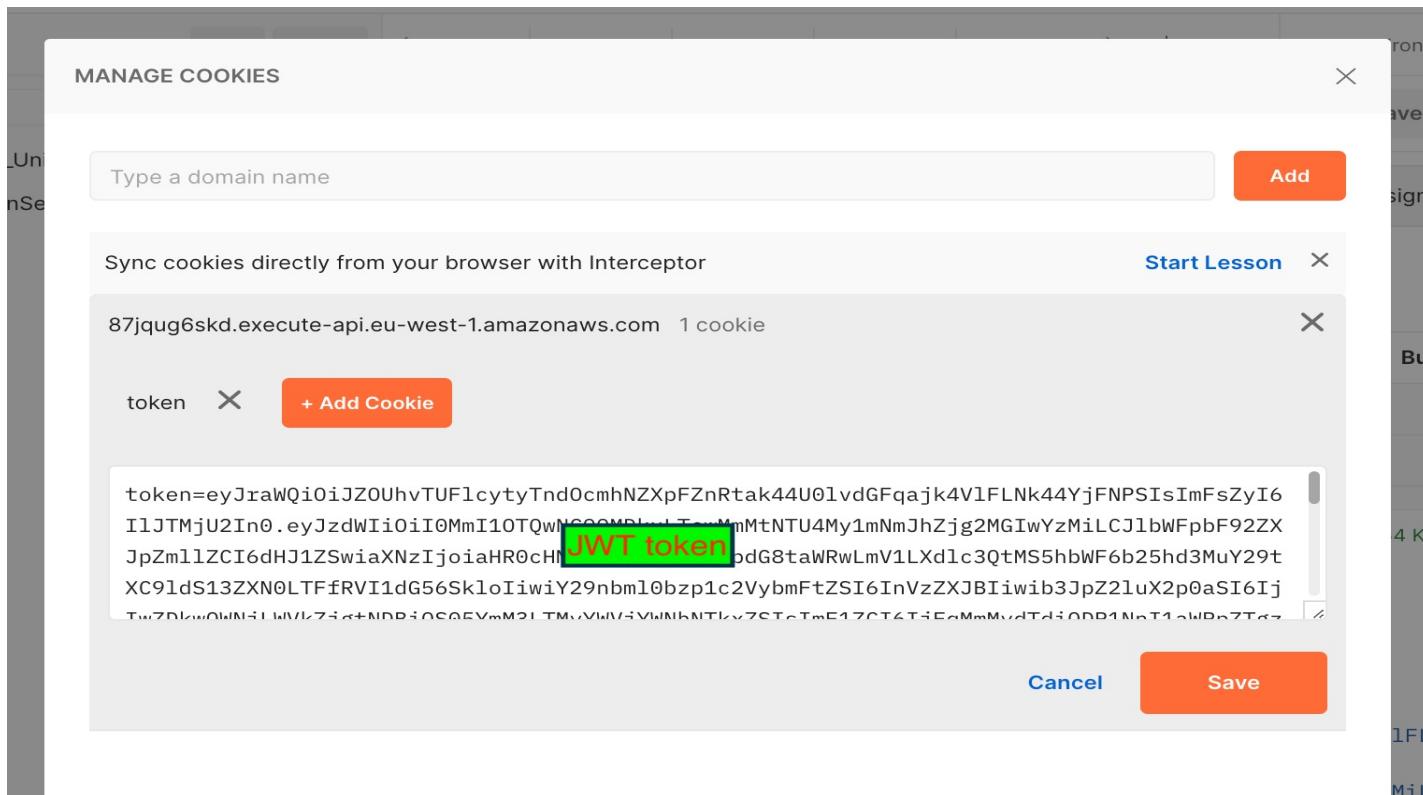


i Restart Visual Studio Code to apply the la

Update Now

# Example – SignIn request / response

# Example – SignIn JWT response



# Example –JWT token decoding

The screenshot shows a Chrome browser window with the jwt.io website loaded. The URL in the address bar is "jwt.io". The page has a pink header with the text "Learn what's new with Auth0 at Devday23 on May 16" and a "Register now →" button. Below the header is the jwt logo and navigation links for "Debugger", "Libraries", "Introduction", and "Ask". To the right, it says "Crafted by auth0 by Okta".

The main area is divided into two sections: "Encoded" on the left and "Decoded" on the right.

**Encoded:** This section contains a large yellow box containing the encoded JWT token:

```
eyJraWQiOiJiejFcL21HV05LVWcxeUNaZVd5eGR
tdkVZSWhRdUlPUlZ1THNyU01YQkF2VT0iLCJhbG
ci0iJSUzI1NiJ9.eyJzdWIiOiIzNTllMjE1Yi0y
ZTA1LTRiN2YtOTEzNS1hMTAxYjhhdNg2MzEiLCJ
1bWFpbF92ZXJpZml1ZCI6dHJ1ZSwiaXNzIjoiaH
R0cHM6XC9cL2NvZ25pdG8taWRwLmV1LXdlc3QtM
S5hbWF6b25hd3MuY29tXC9ldS13ZXN0LTFFchRH
WE9IM1VyIwiY29nbml0bzp1c2VybmbFtZSI6InV
zZXJBIiwib3JpZ2luX2p0aSI6IjIwM2UwMDnhLT
E4YWItnGRjOS1i0DU3LTfKOGNiNjMzNWZkNiIsI
mF1ZCI6IjM2M2NycGFoMmdhbDRmY2JsbDFyZGN1
YmIxIwiZXlbnRfaWQi0iI4MWY5NmY3ZS1hMmF
kLTQ2N2EtYmVjMy0zODY2M2NlOTlmZTMiLCJ0b2
t1b191c2Ui0iJpZCIsImF1dGhfdGltZSI6MTY4N
zI2NTg2NiwiXhwIjoxNjg3MjY5NDY2LCJpYXQi
0jE20DcyNjU4NjYsImp0aSI6IjY3N2Y10WU5Ltc
xNDUtNDczMy05NjI3LTK1MmVhYTliZDFmNSIsIm
```

**Decoded:** This section shows the decoded JWT token with its header and payload.

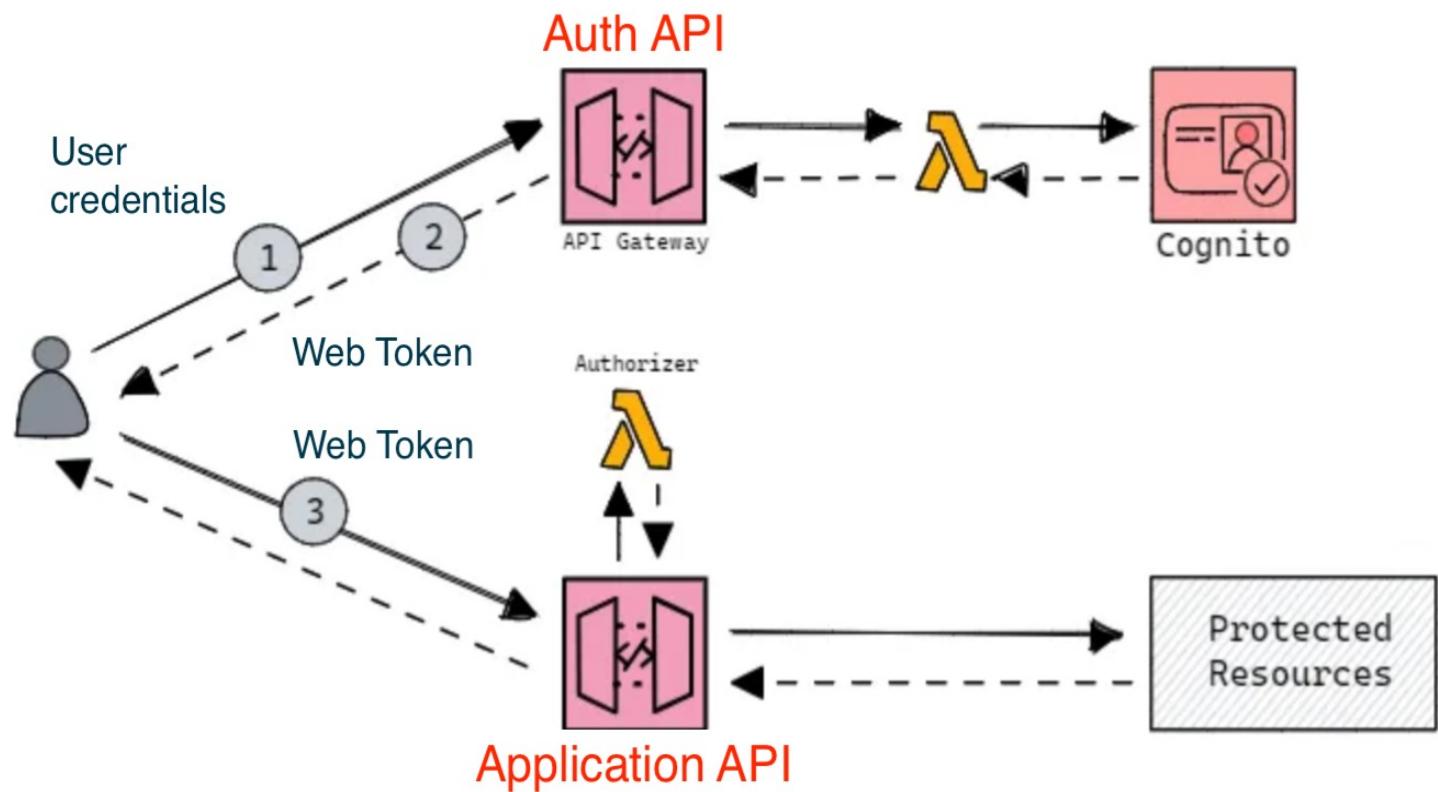
**HEADER: ALGORITHM & TOKEN TYPE**

```
"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.ptGX0H3Ur",
  "cognito:username": "userA",
  "origin_jti": "203e003a-18ab-4dc9-b857-1d8cb6335fd6",
  "aud": "363crpah2gal4fcbl11rdcubb1",
  "event_id": "81f96f7e-a2ad-467a-bec3-38663ce99fe3",
  "token_use": "id",
  "auth_time": 1687265866,
  "exp": 1687269466,
  "iat": 1687265866,
  "jti": "677f59e9-7145-4733-9627-952ea9bd1f5",
  "email": "oconnordiarmuid@gmail.com"
```

# Example



# Example – App API infrastructure

```
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));
```

The code snippet shows the configuration of an API. It starts by creating an API object and defining two resources: 'protected' and 'public'. Each resource has a corresponding function: 'protectedFn' and 'publicFn'. An 'authorizerFn' is also defined. A 'requestAuthorizer' is created with specific configuration: it uses a cookie header for identity sources, delegates to the 'authorizerFn' as the handler, and caches results for zero minutes. This 'requestAuthorizer' is then used in the 'addMethod' calls for both the 'protected' and 'public' resources, indicating that requests to these endpoints require custom authorization handling.

# Example – Protected route request

The screenshot shows a POST request in Postman to the URL `https://nt6d5yy9wc.execute-api.eu-west-1.amazonaws.com/prod/protected`. The request has the following headers:

- Host**: `iDiJiejFcL21HV055eGRtdkVZSWhR01YQkF2VT0iLCJNiu9.evJzdWliOilzN` (highlighted with a red circle)
- User-Agent**: `PostmanRuntime/7.28.4`
- Accept**: `/*`
- Accept-Encoding**: `gzip, deflate, br`
- Connection**: `keep-alive`
- Cookie**: `token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIaWE1F...IUpUIZ1` (highlighted with a red circle)

The response status is `200 OK` with a response time of `811 ms` and a response size of `350 B`. The response body is:

```
1 You received a super secret!!
```

Annotations in the screenshot include:

- A red arrow points from the `Host` header to the value in the Headers table.
- A red arrow points from the `Cookie` header to the value in the Headers table.
- A red arrow points from the `Host` header to the URL bar above.
- A red arrow points from the `Cookie` header to the `Body` tab below.
- A red arrow points from the `Cookie` value in the Headers table to the `Body` tab below.
- A red arrow points from the `Body` tab to the response body.

# Example –Request Authorizer

- Steps performed by authorizer:
  1. Parse value of HTTP Request Cookie header into a Map data structure for convenience.
  2. Find ‘token’ key in Map.
  3. If it’s missing,
    - Return an IAM policy Denying use of the app 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 API.  
Else
    - Return policy Denying execution of the app API.

