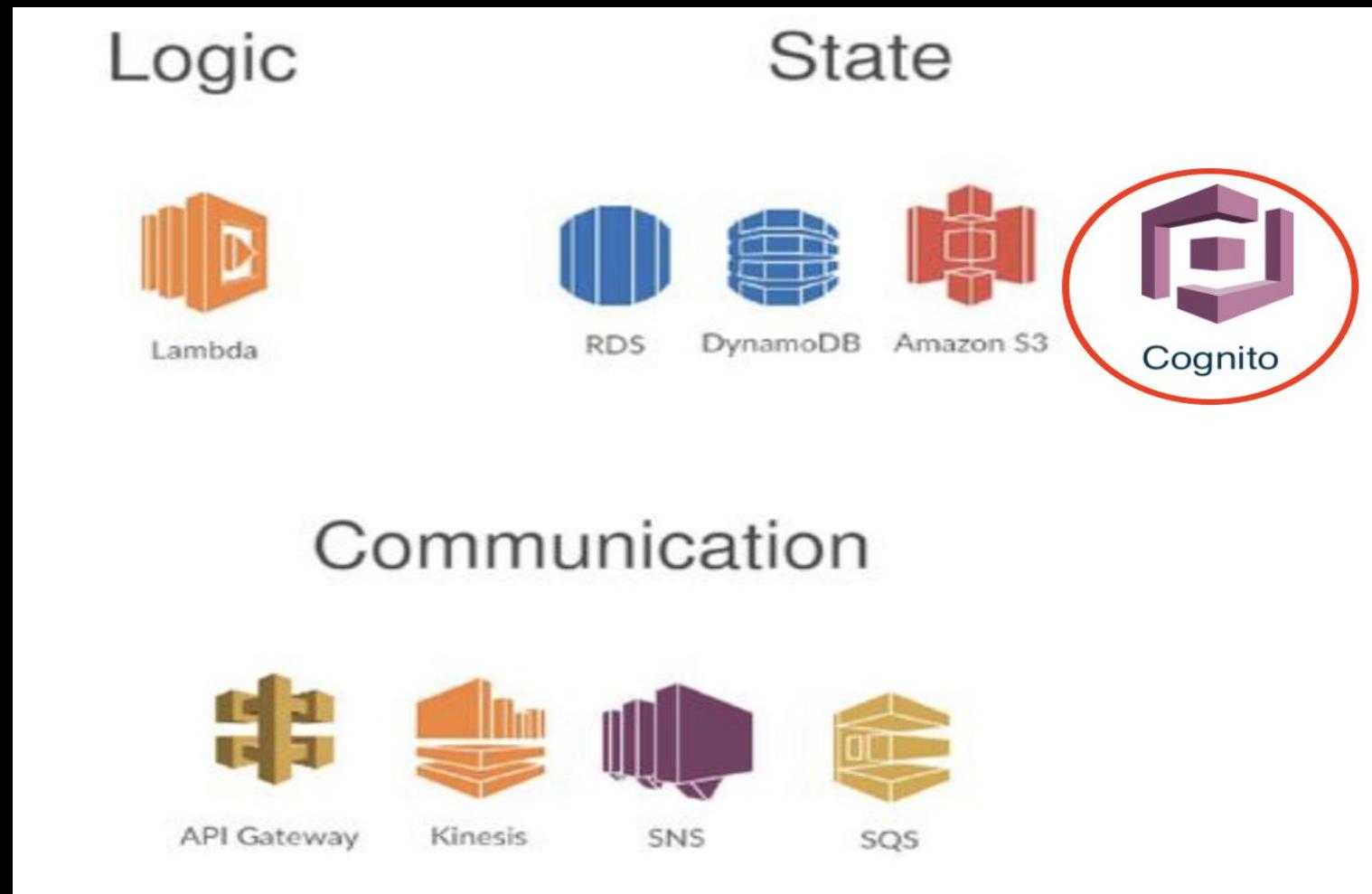


# Amazon Cognito

# Components of a Serverless app



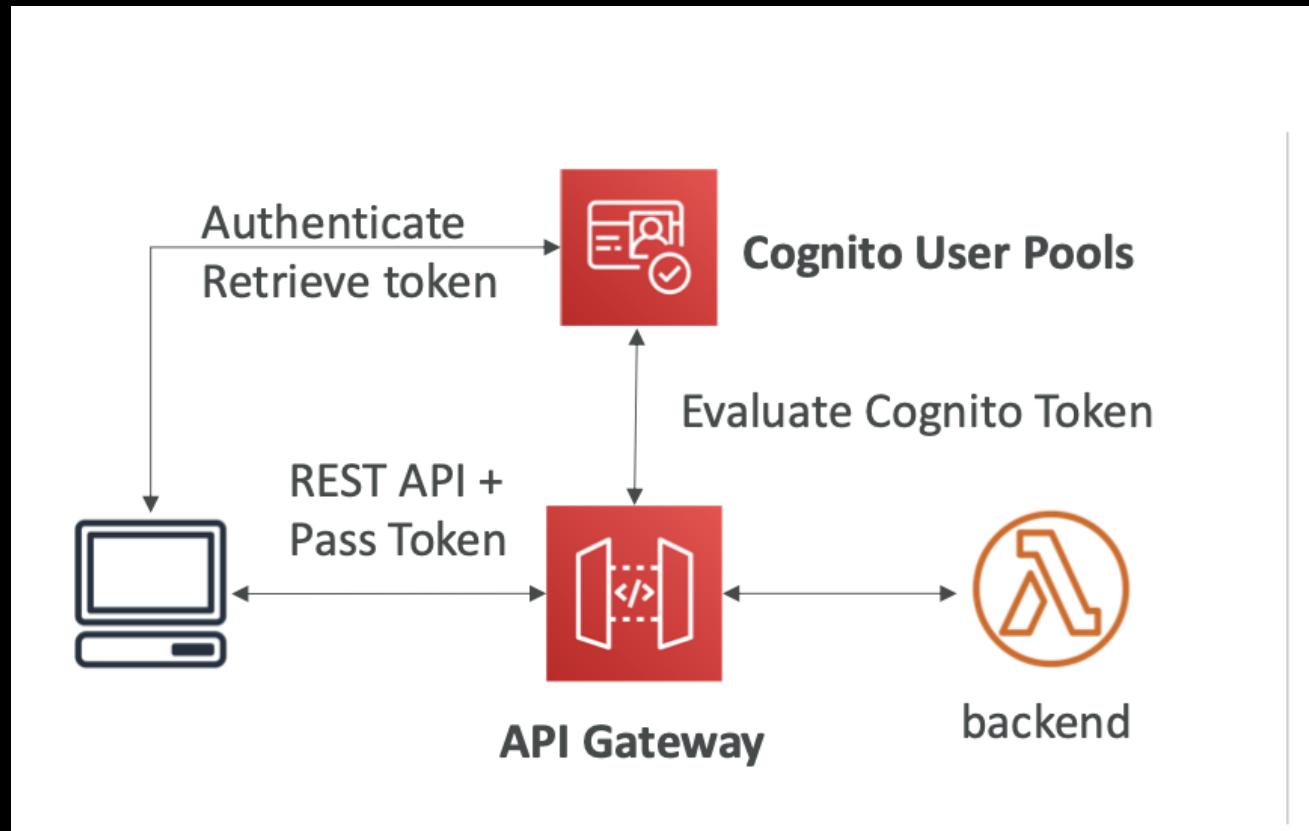
# Amazon Cognito

- 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”.

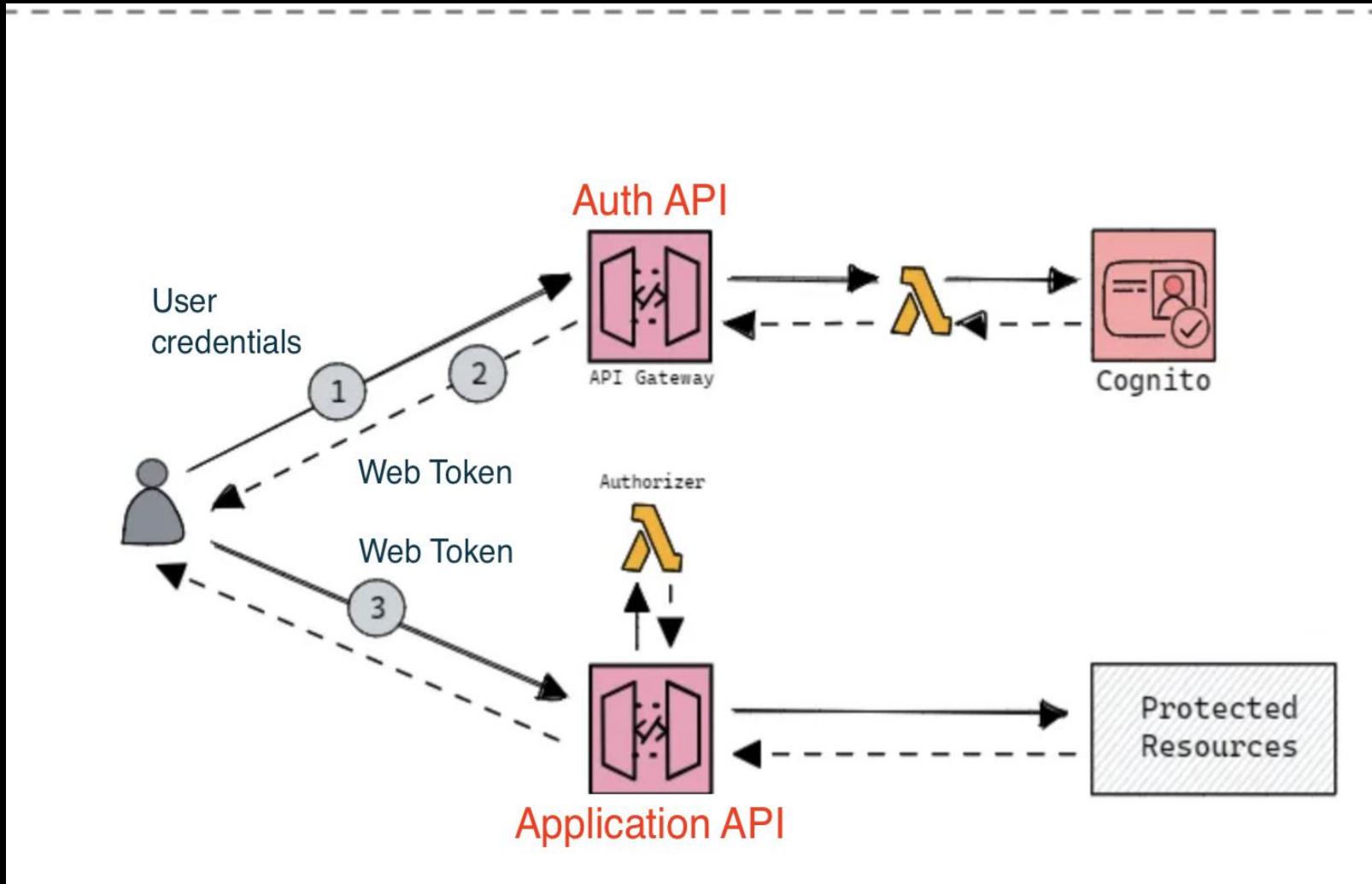
# Cognito User Pool.

- 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's role

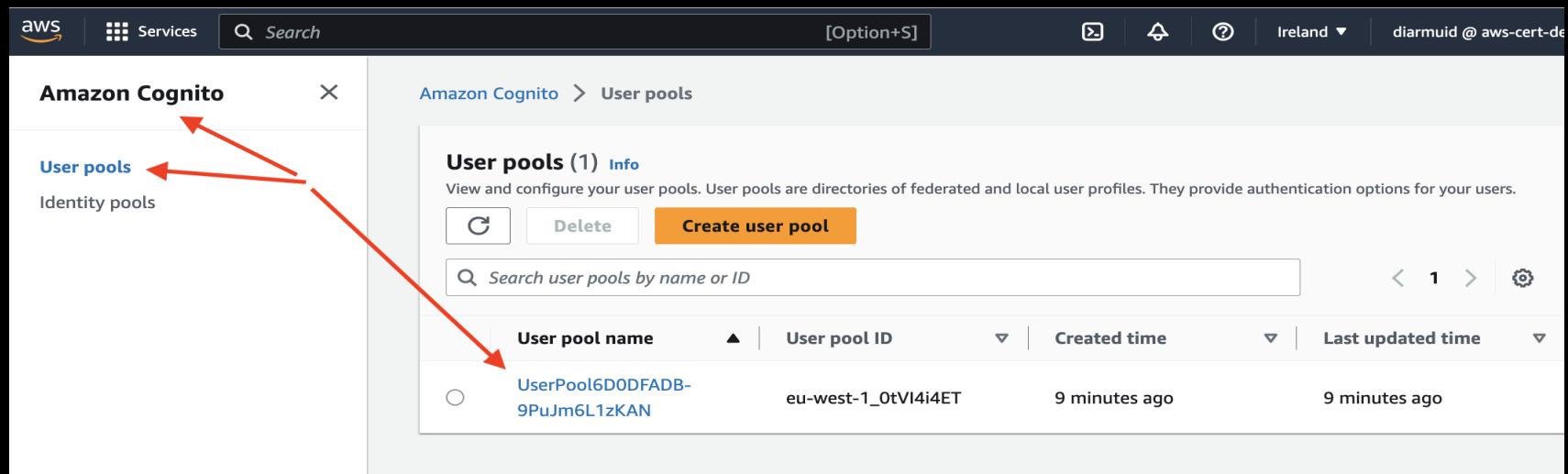


# Typical architecture

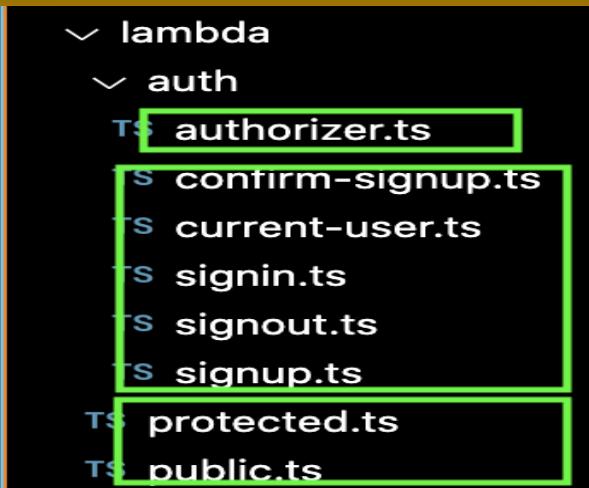
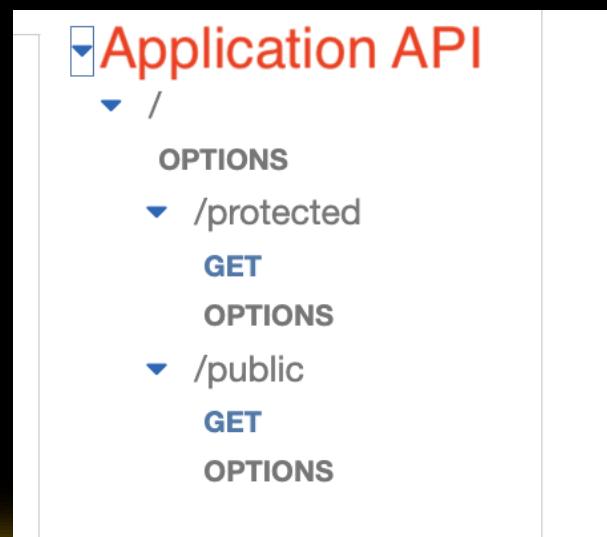
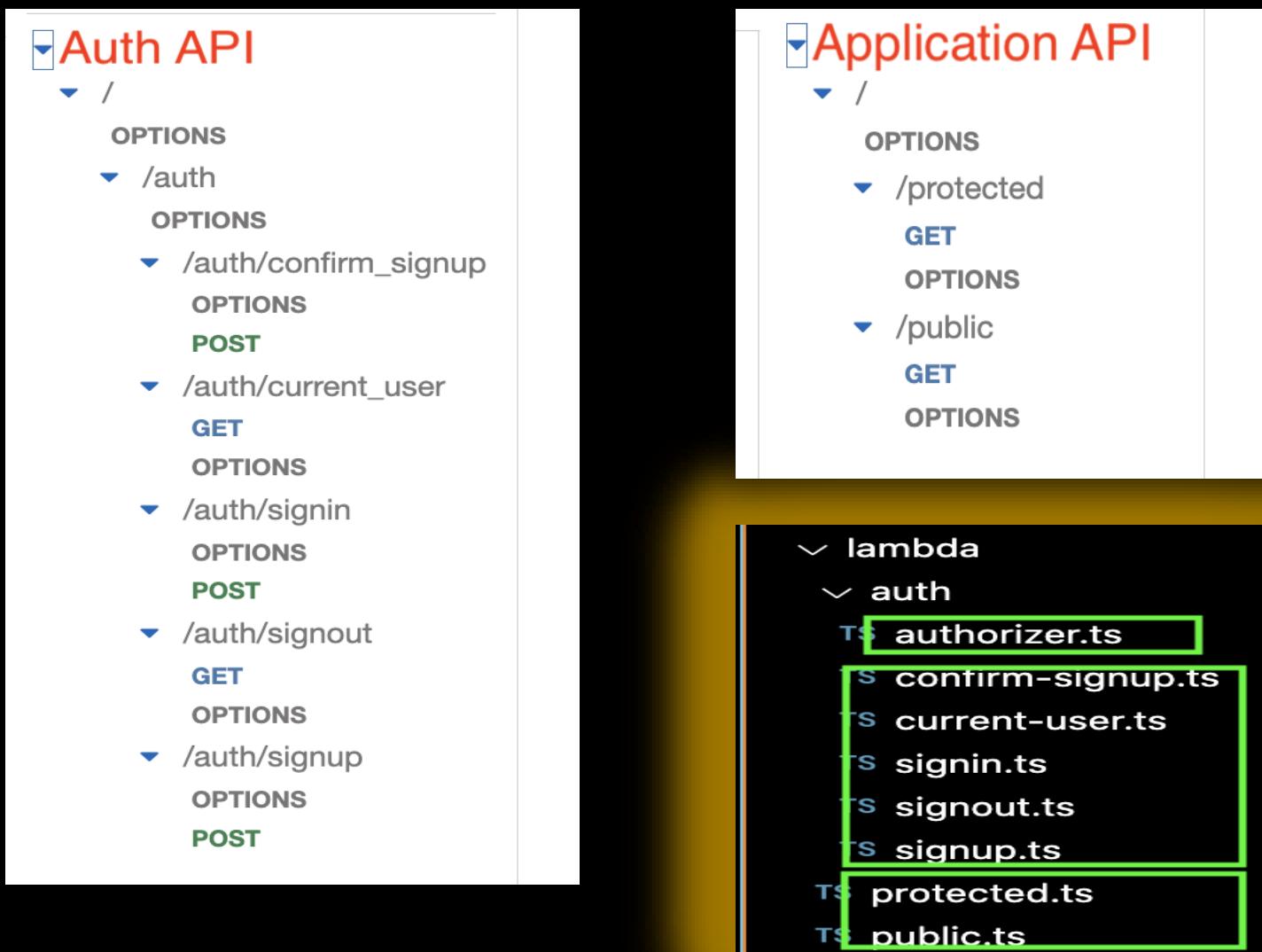


# Demo – User pool

```
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
```



# Demo – APIs.



# Demo – Sign up

The screenshot shows a Postman interface with the following details:

- Method:** POST
- URL:** <https://87jqug6skd.execute-api.eu-west-1.amazonaws.com/prod/auth/signu...>
- Body (JSON):**

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

An orange arrow points to the password field.
- Response Status:** 200 OK
- Response Time:** 1630 ms
- Response Size:** 613 B
- Response Content (Pretty):**

```
1 {
2   "message": {
3     "$metadata": {
4       "httpStatusCode": 200,
5       "requestId": "ad8071ef-a53a-4df9-8b0c-c96fb5631bbe",
6       "attempts": 1,
```

# Demo – SignUp

```
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  };

```

# Demo – Confirm SignUp

The screenshot shows a Postman interface with the following details:

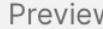
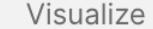
- Method:** POST
- URL:** `https://87jqug6skd.execute-api.eu-west-1.amazonaws.com/prod/auth/confirm_signup`
- Headers:** (9)
- Body:** (Green dot icon) - Selected tab.
- Params:**
- Auth:**
- Pre-req.:**
- Tests:** (Orange arrow points here)
- Settings:**
- Cookies:**

The Body section is set to "Text" and contains the following JSON payload:

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

An orange arrow points from the "Tests" tab to the "code" field in the JSON body.

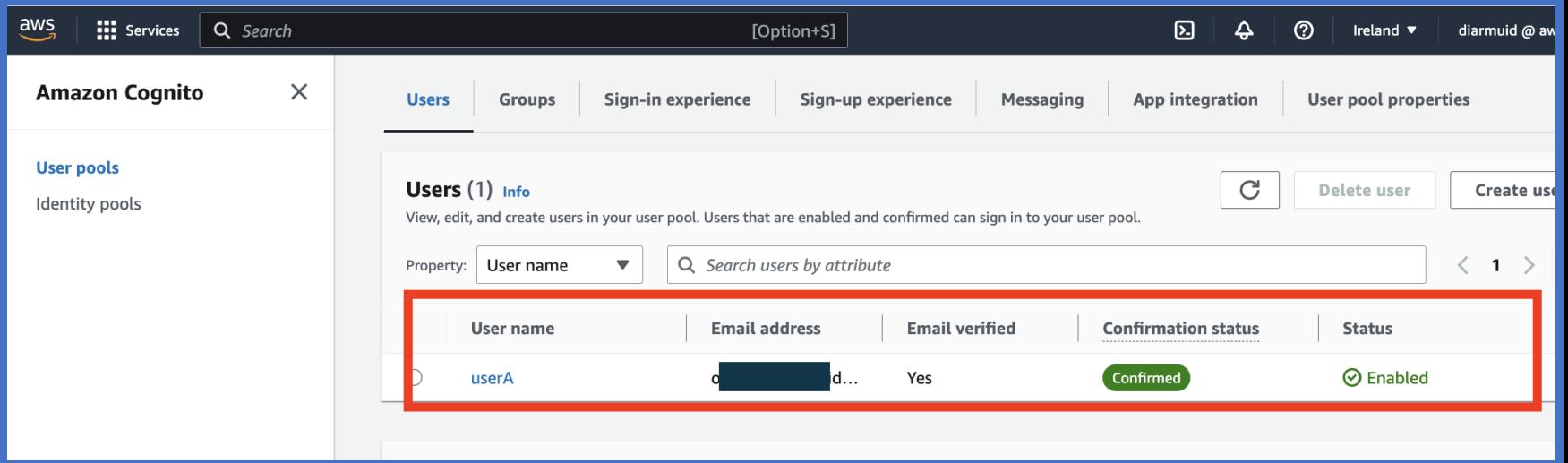
Below the Body section, the response is displayed:

Body       

200 OK 864 ms 374 B Save Response  

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

# Demo – Confirm SignUp



The screenshot shows the AWS Cognito User Pools console. The left sidebar has 'User pools' selected. The main 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' shows 'userA' in the 'User name' column, an email address starting with 'o...' in the 'Email address' column, 'Yes' in 'Email verified', 'Confirmed' in 'Confirmation status', and 'Enabled' with a checked checkbox in 'Status'. A red box highlights the entire row for 'userA'.

User name	Email address	Email verified	Confirmation status	Status
userA	o...d...	Yes	Confirmed	<input checked="" type="checkbox"/> Enabled

# Demo – 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) { ... }
183 };
184
```

# Demo – Sign In

```
186 const client = new CognitoIdentityProviderClient({ region: "eu-west-1" });
187
188 export const handler: APIGatewayProxyHandlerV2 = async (event) => {
189   const { username, password } = JSON.parse(event.body);
190   const params: InitiateAuthCommandInput = {
191     ClientId: process.env.CLIENT_ID!,
192     AuthFlow: "USER_PASSWORD_AUTH",
193     AuthParameters: {
194       USERNAME: username,
195       PASSWORD: password,
196     },
197   };
198   try {
199     const command = new InitiateAuthCommand(params);
200     const { AuthenticationResult } = await client.send(command);
201     const token = AuthenticationResult.IdToken;
202
203     return {
204       statusCode: 200,
205       headers: {
206         "Access-Control-Allow-Headers": "*",
207         "Access-Control-Allow-Origin": "*",
208         "Set-Cookie": `token=${token}; SameSite=None; Secure; HttpOnly; Path=/; Max-Age=31536000;`,
209       },
210       body: JSON.stringify({
211         message: "Auth successfull",
212         token: token,
213       }),
214     };
215   } catch (err) {.... }
216 };
217 }
```

 Restart Visual Studio Code to apply the language pack changes.

[Update Now](#)

# Demo – Sign In request / response

The screenshot shows a POST request to `https://87jqug6skd.execute-api.eu-west-1.amazonaws.com/prod/auth/signin`. The request body contains the following JSON:

```
1 {  
2   "username": "userA",  
3   "password": "passABCDE!2"  
4 }  
5
```

A red arrow points from a green callout box to the `Cookies` tab in the top navigation bar. The callout box contains the text: **Click to see list of cookies sent by API**.

The response status is 200 OK, with a response time of 1971 ms and a size of 2.54 KB. The response body is:

```
1 {  
2   "message": "Auth successfull",  
3   "token":  
4     "eyJraWQiOiJZ0UhvTUFlcytyTnd0cmhNZXpFZnRtak44U01vdGFqajk4V1FLNk44YjFNPSIsImFs  
5 ZyI6IlJTMyU2In0.  
6 eyJzdWIiOiI0MmI10TQwNC00MDkxLTcwMmMtNTU4My1mNmJhZjg2MGIwYzMlCJ1bWFpbF92ZXJpZ  
7 m11ZCI6dHJ1ZSwiaXNzIjoiaHR0cHM6XC9cL2NvZ25pdG8taWRwLmV1LXdlc3QtMS5hbWF6b25hd3  
8 MuY29tXC91dS13ZXN0LTFFrVI1dG56SkloIiwiY29nbml0bzp1c2VybmtZSI6InVzZXJBIiwib3J  
9 -701...Y29nbml0bzp1c2VybmtZSI6InVzZXJBIiwib3J
```

Red arrows point to the token value in the response body and to the first few characters of the token string.

# Demo – Sign In JWT token

MANAGE COOKIES

Type a domain name  Add

Sync cookies directly from your browser with Interceptor [Start Lesson](#)

87jqug6skd.execute-api.eu-west-1.amazonaws.com 1 cookie

token X [+ Add Cookie](#)

token=eyJraWQiOiJZOUhvTUF1cytyTnd0cmhNZXpFZnRtak44U01vdGFqajk4V1FLNk44YjFNPSIsImFsZyI6I1JTMjU2In0.eyJzdWIiOiI0MmI10TQwNC00MDIwLTgwMmMtNTU4My1mNmJhZjg2MGIwYzMlCJ1bWFpbF92ZXJpZml1ZCI6dHJ1ZSwiaXNzIjoiaHR0cHM6dG8taWRwLmV1LXd1c3QtMS5hbWF6b25hd3MuY29tXC91dS13ZXN0LTFFRVI1dG56SkloIiwiY29nbml0bzp1c2VybmFtZSI6InVzZXJBIiwib3JpZ2luX2p0aSI6IjTmZDkxOQm4IiwiIz4g7NDR40SAEVmM9I TMvVIVV4VMMNNTIv7CTcTmE17CT6T4E6MmMvdtdd0D1NpT1owBnZT6>

[Cancel](#) [Save](#)

# JWT token decoding

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

```
eyJraWQiOiJiejFcL21HV05LVWcxeUNaZVd5eGR  
tdkVZSWhRdUlPUlZ1THNyU01YQkF2VT0iLCJhbG  
ci0iJSUzI1NiJ9.eyJzdWIiOiIzNTl1MjE1Yi0y  
ZTA1LTRiN2YtOTEzNS1hMTAxYjhhdg2MzEiLCJ  
1bWFpbF92ZXJpZml1ZCI6dHJ1ZSwiaXNzIjoiaH  
R0cHM6XC9cL2NvZ25pdG8taWRwLmV1LXdlc3QtM  
S5hbWF6b25hd3MuY29tXC9ldS13ZXN0LTFFfcHRH  
WE9IM1VyIiwiY29nbml0bzp1c2VybmrZSI6InV  
zZXJBIiwib3JpZ2luX2p0aSI6IjIwM2UwMDNhLT  
E4YWIItNGRjOS1iODU3LTfKOGNiNjMzNWZkNiIsI  
mF1ZCI6IjM2M2NycGFoMmdhbDRmY2JsbDFyZGN1  
YmIxIiwiZXZ1bnRfaWQiOiI4MWY5NmY3ZS1hMmF  
kLTQ2N2EtYmVjMy0zODY2M2N10TlmZTMiLCJ0b2  
t1b191c2Ui0iJpZCIsImF1dGhfdGltZSI6MTY4N  
zI2NTg2NiwiZXhwIjoxNjg3MjY5NDY2LCJpYXQi  
0jE20DcyNjU4NjYsImp0aSI6IjY3N2Y10WU5Ltc  
xNDUtNDczMy05NjI3LTk1MmVhYTl1ZDFmNSIsIm
```

**Decoded** EDIT THE PAYLOAD AND SECRET

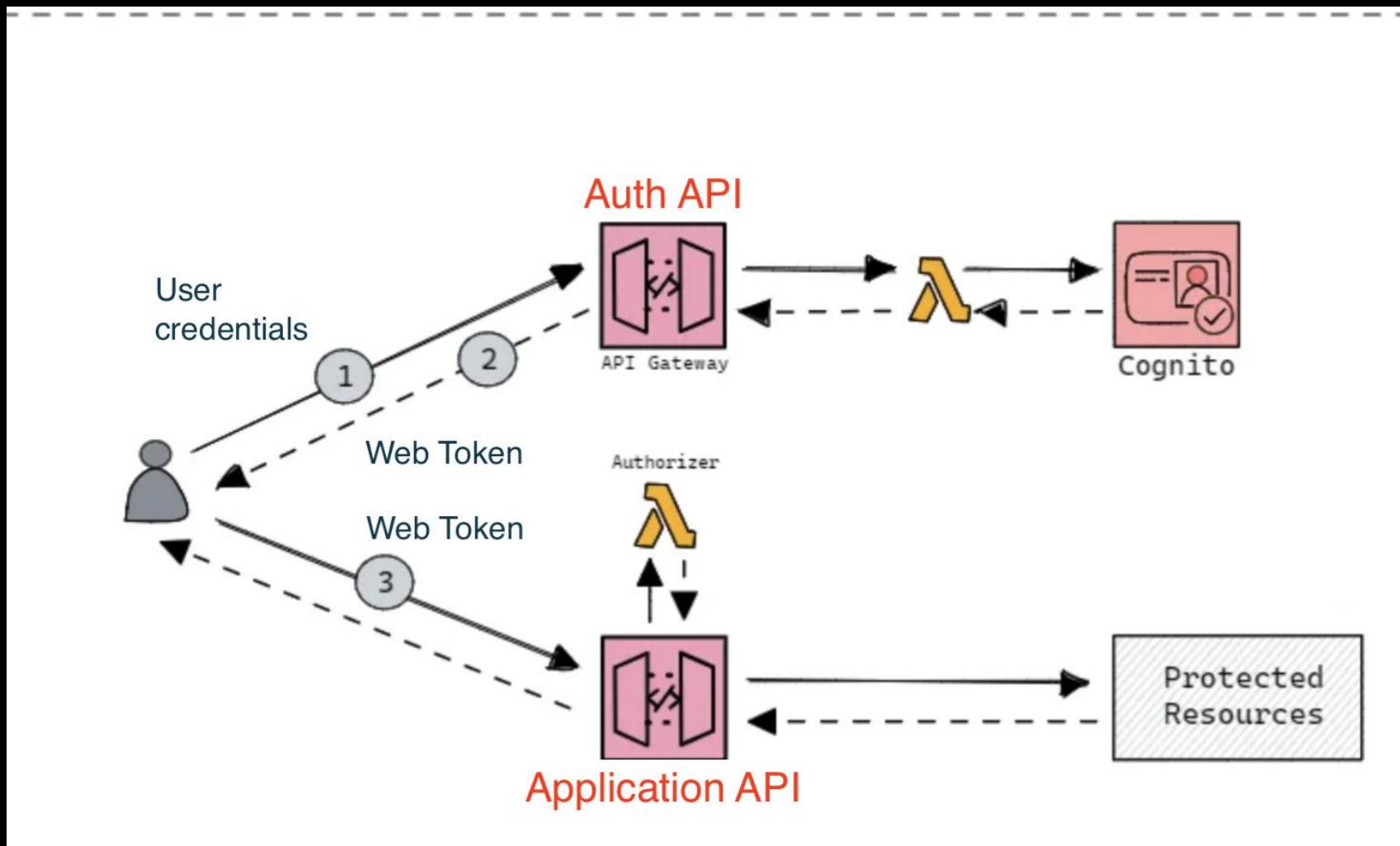
HEADER: ALGORITHM & TOKEN TYPE

```
"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": "363crpah2gal4fcbl11rdcubb1",  
  "event_id": "81f96f7e-a2ad-467a-bec3-38663ce99fe3",  
  "token_use": "id",  
  "auth_time": 1687265866,  
  "exp": 1687269466,  
  "iat": 1687265866,  
  "jti": "677f59e9-7145-4733-9627-952eaa9bd1f5",  
  "email": "oconnordiaruid@gmail.com"
```

# Demo - Architecture



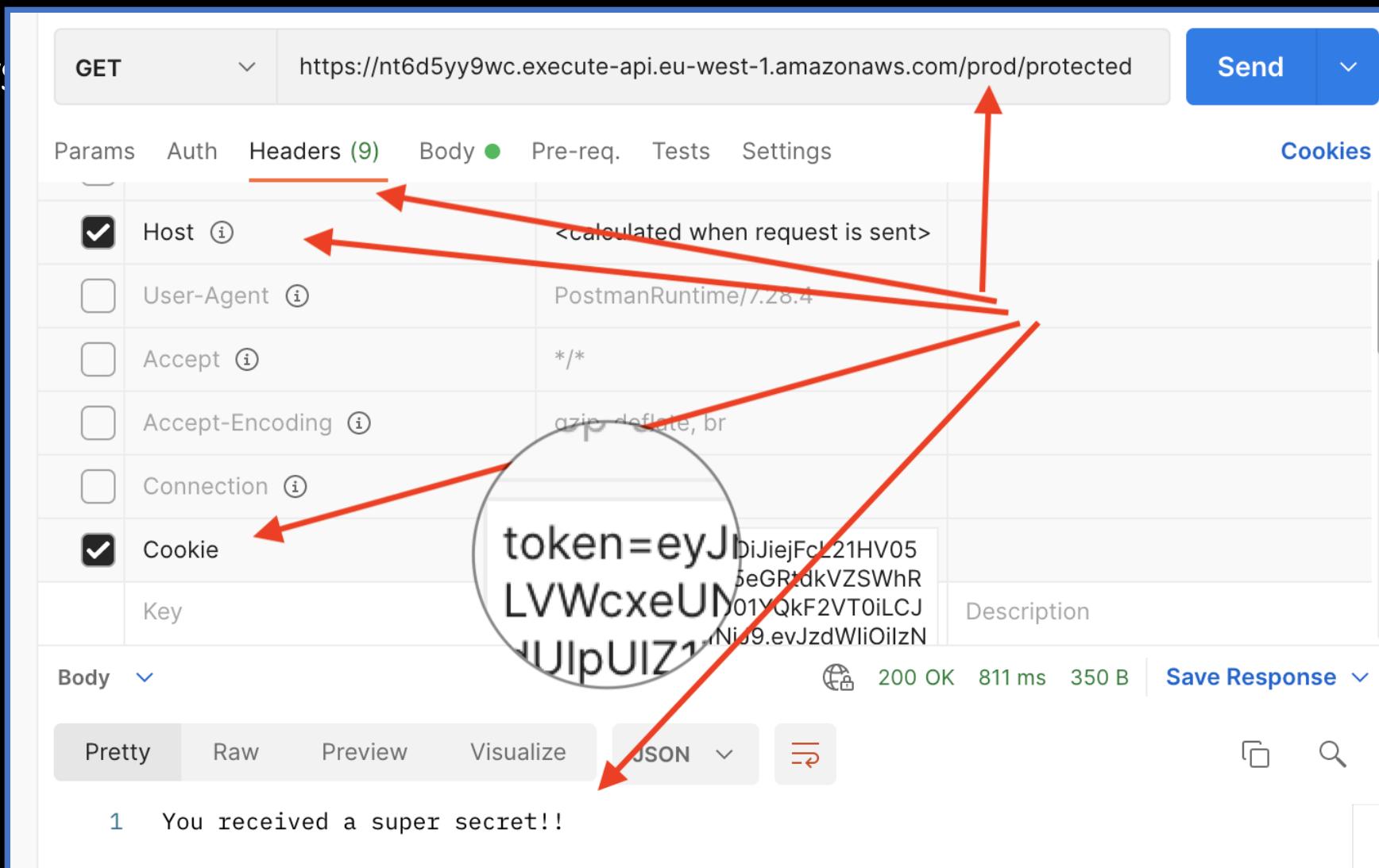
# Demo – App API infrastructure.

```
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));
252
```



# Demo – Protected route HTTP request

- For AWS



# Demo – Path parameters

- Steps performed by authorizer:
  1. Parse HTTP Request Cookie header value & store in a local Map data structure.
  2. Find ‘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.

# API Keys & Usage plans

- API keys are alphanumeric string values that you distribute to application developers to grant access to your API.
  - Keys can be generated by the API Gateway service, or imported into API Gateway from an external source.
- A usage plan specifies who can access one or more deployed API stages (e.g. dev, prod) methods, and optionally sets the target request rate to start throttling requests.
  - A plan uses API keys to identify the clients, and determine their access rights to associated APIs.

# API Keys & Usage plans

