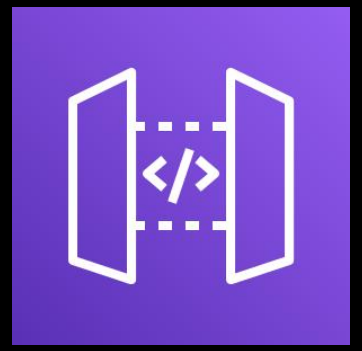
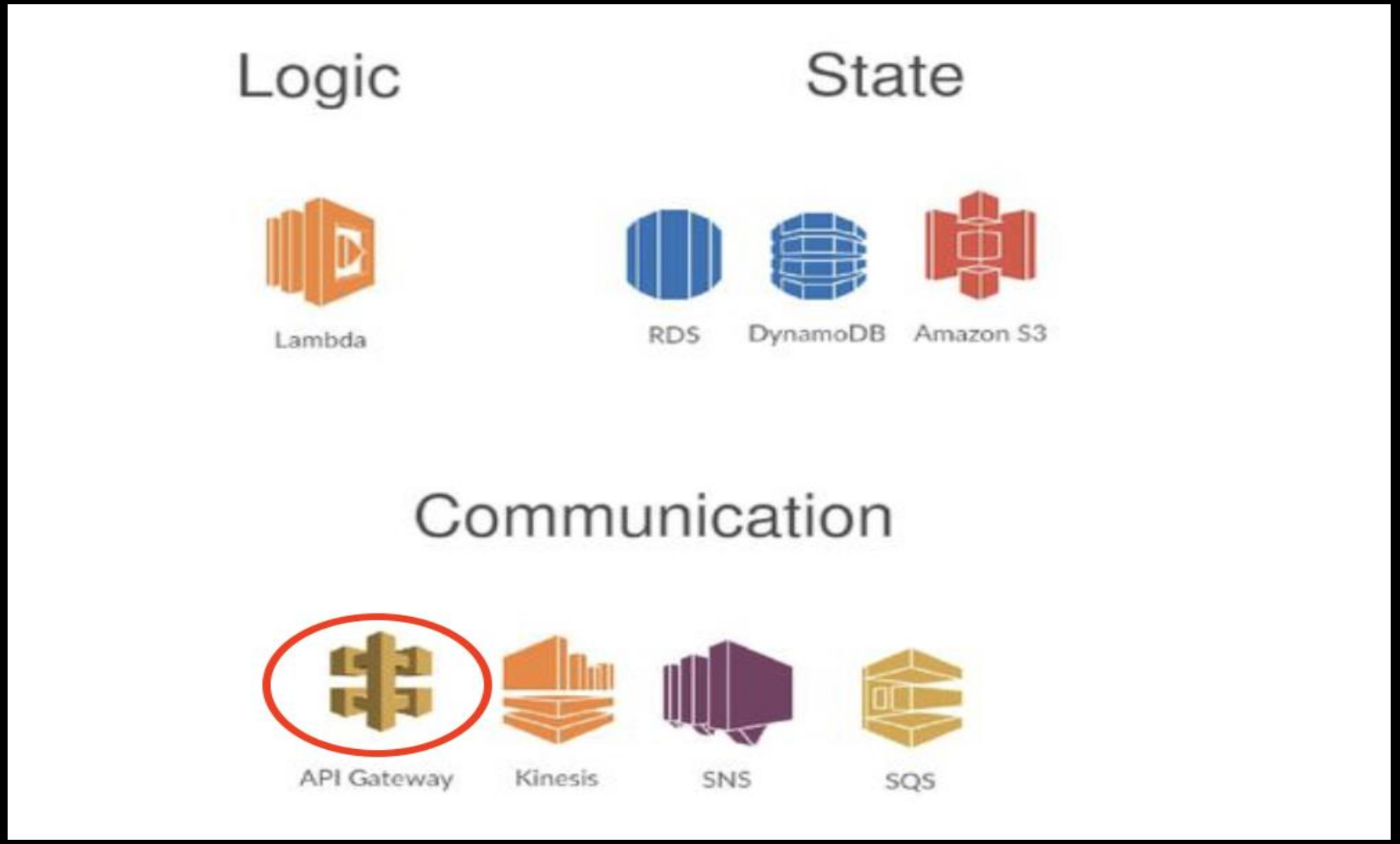
Serverless Web APIs (Contd.)



API Gateway

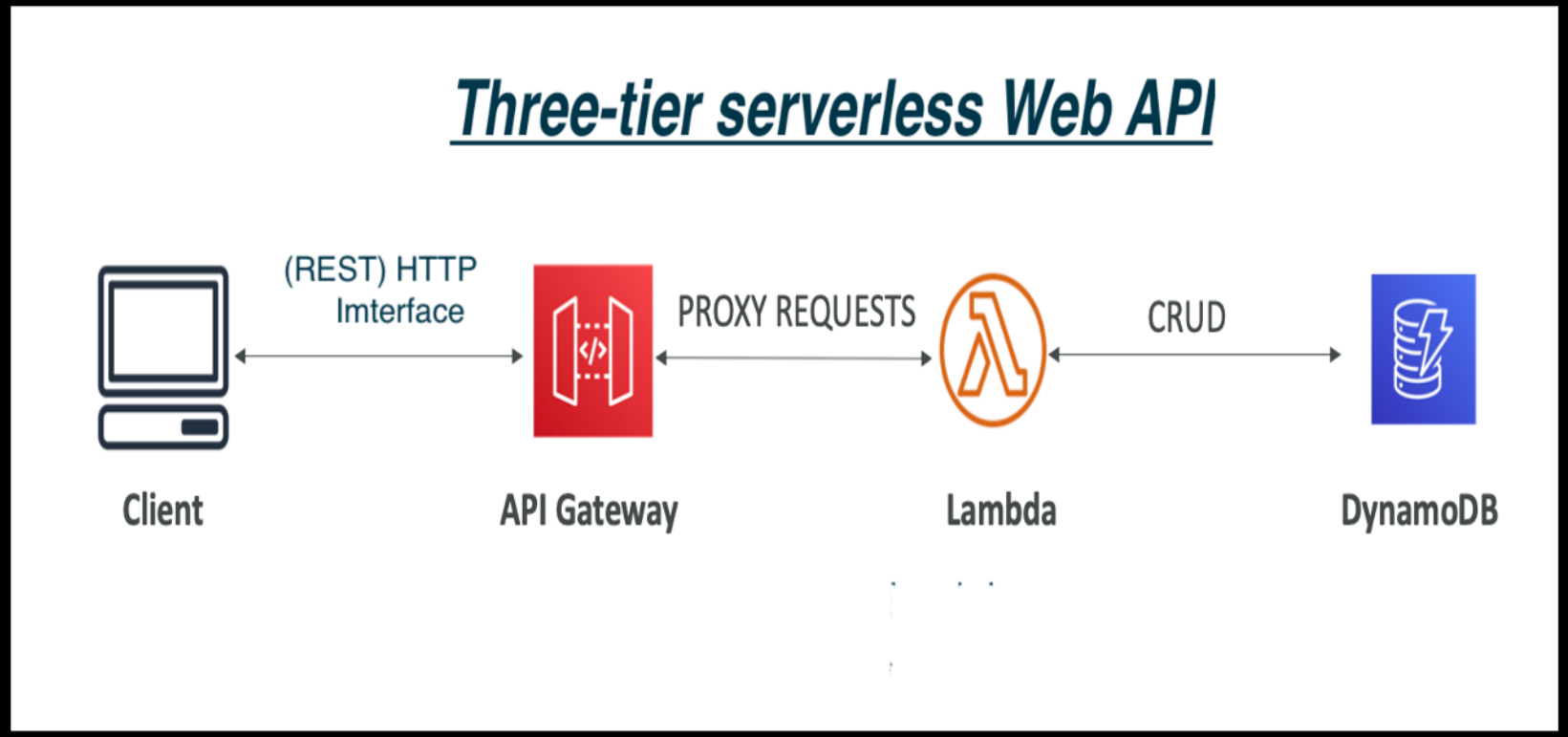
HTTP & REST Web APIs

Components of a Serverless app.



API Gateway

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



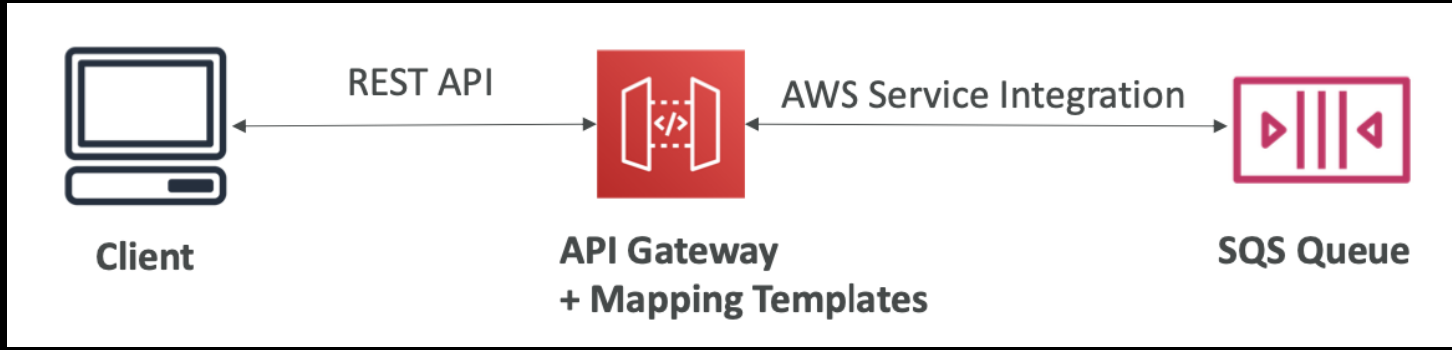
Features

- 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.

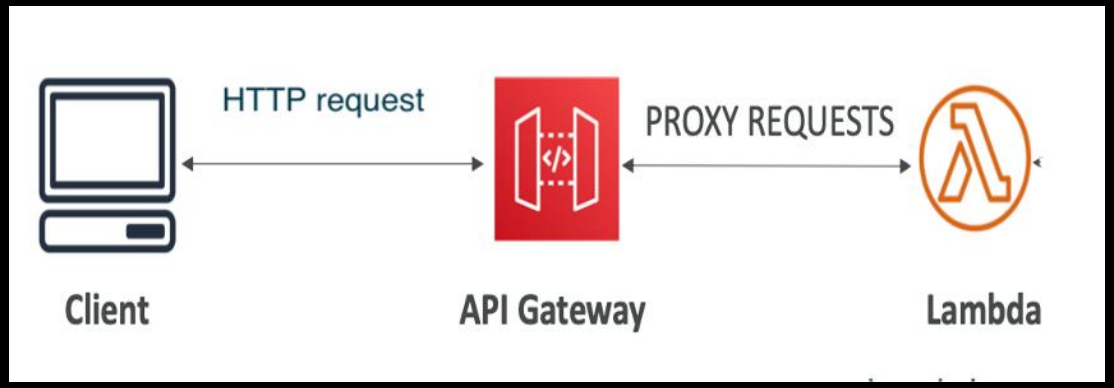
- MOCK Integration Type:
  - API Gateway returns a response without sending the request to the backend.

- 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.



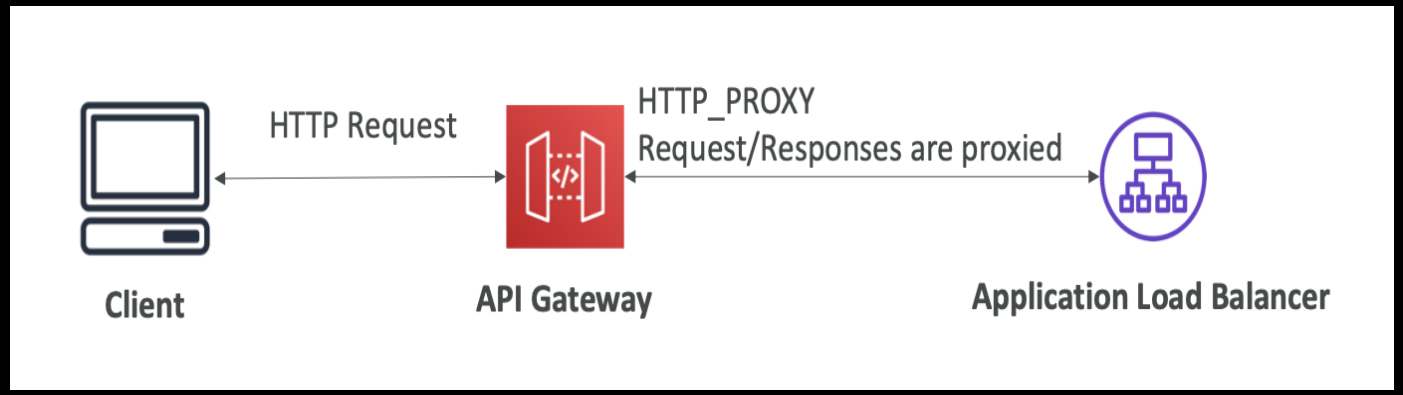
Integration Types.

- 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.



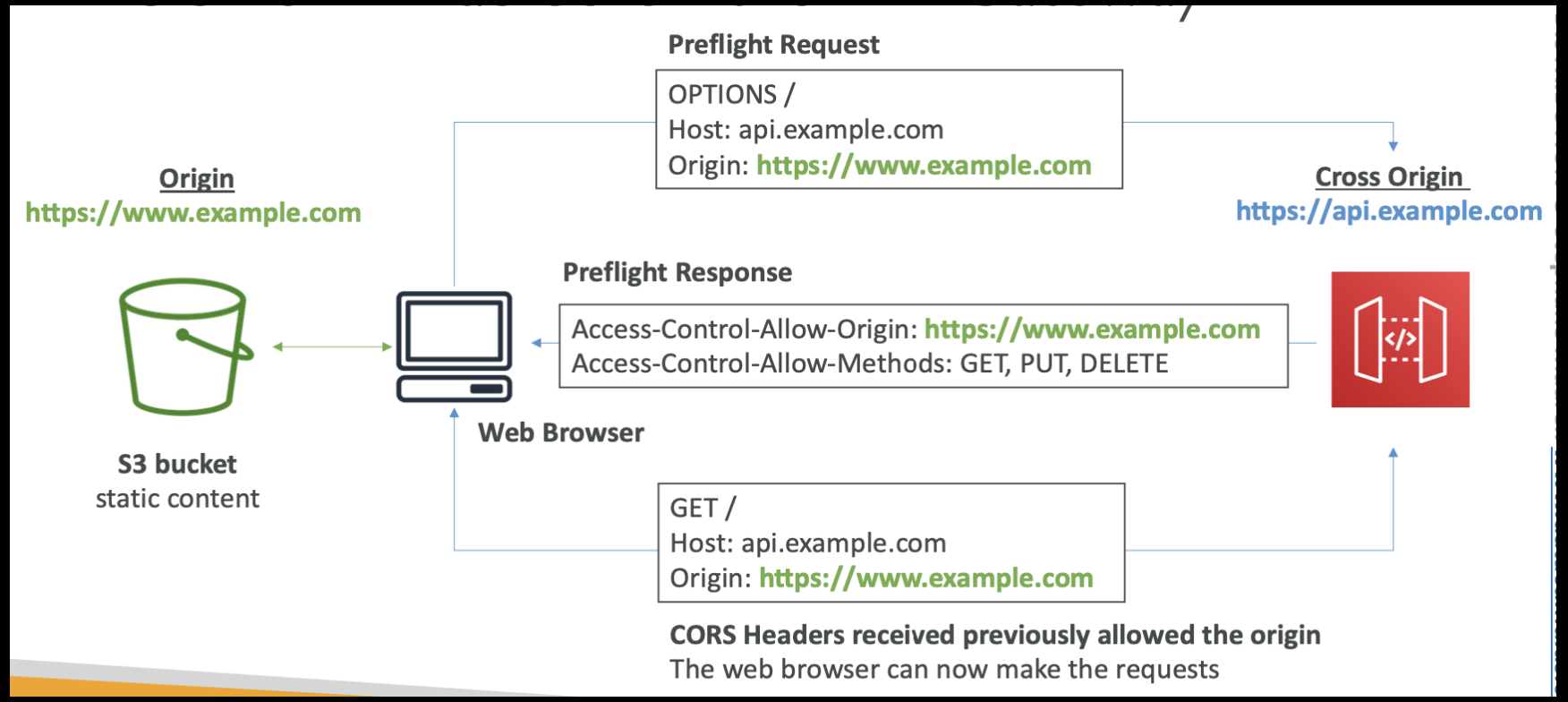
Integration Types.

- 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)

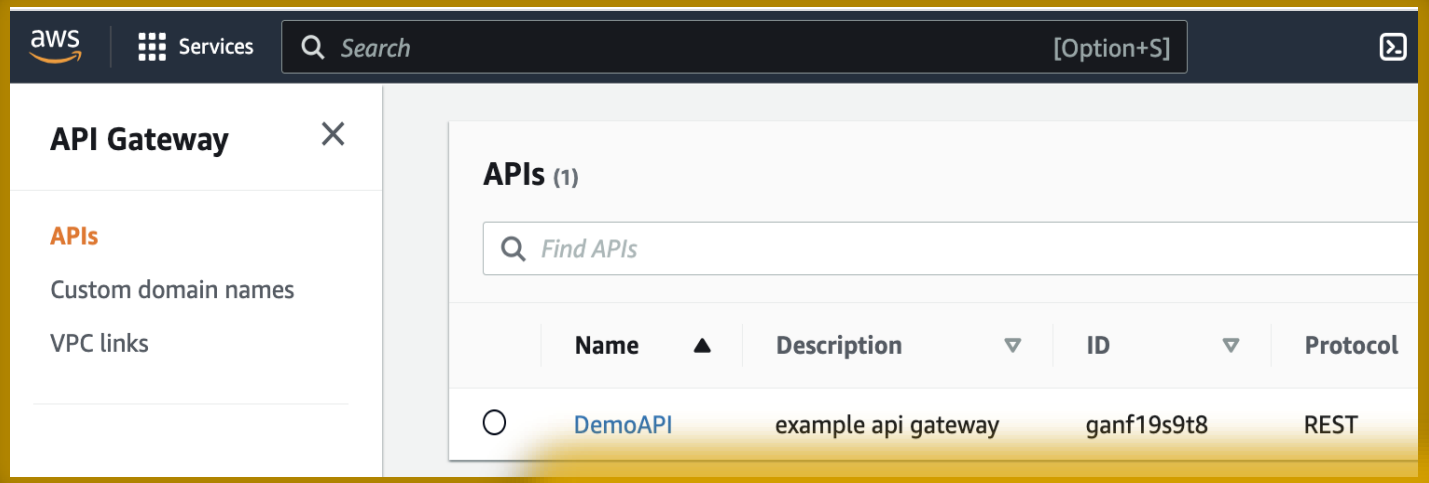
- CORS must be enabled when you receive API calls from another domain.

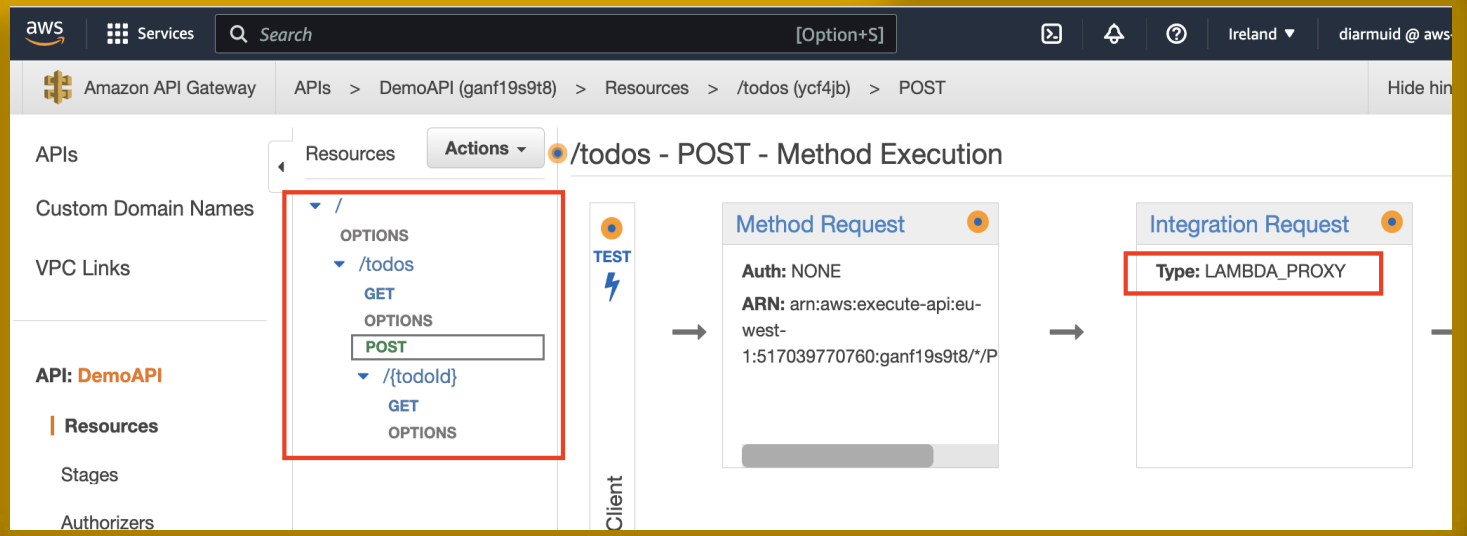


Sample

- 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





```
// 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 })
);
```

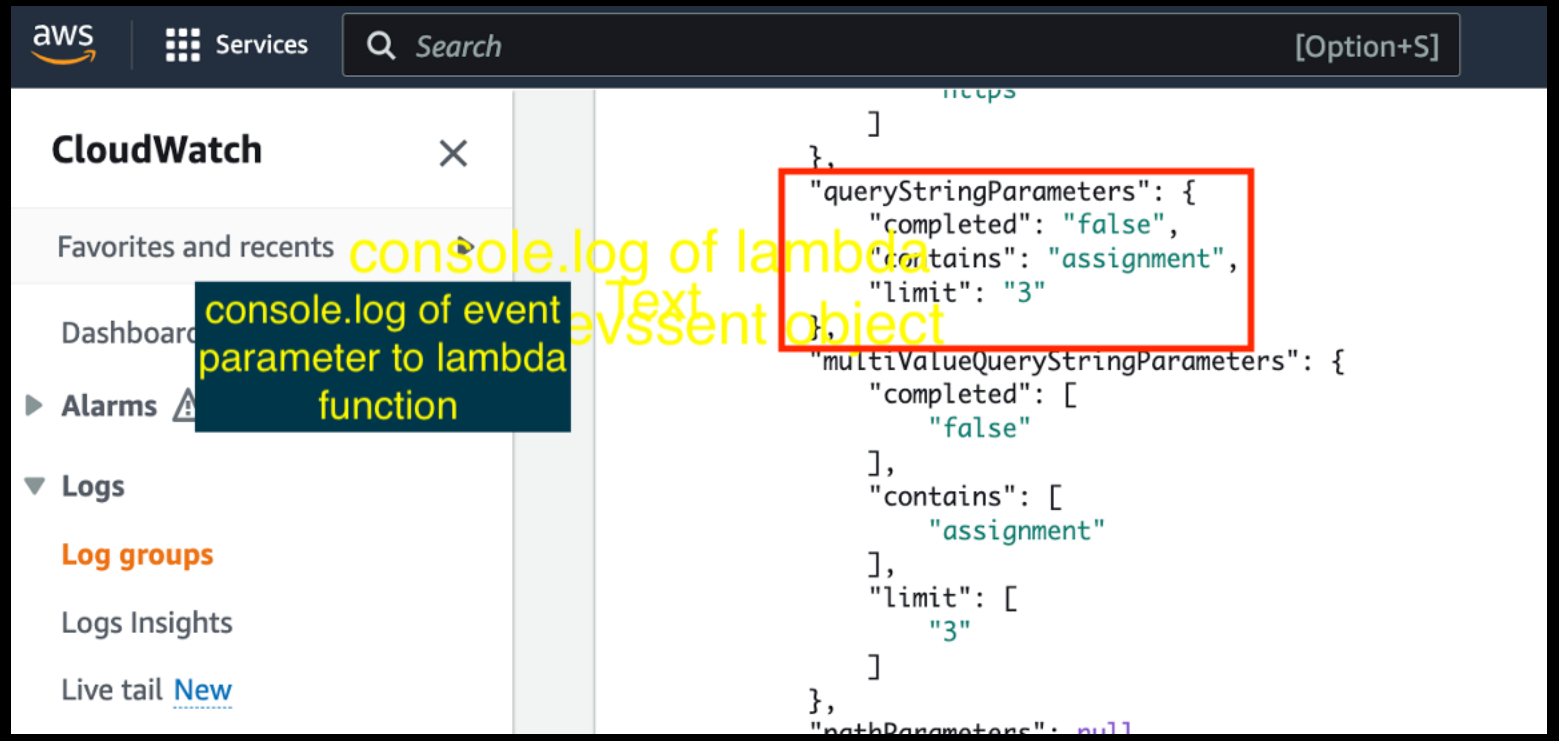
CDK stack to create REST Web API, using API Gateway service

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?

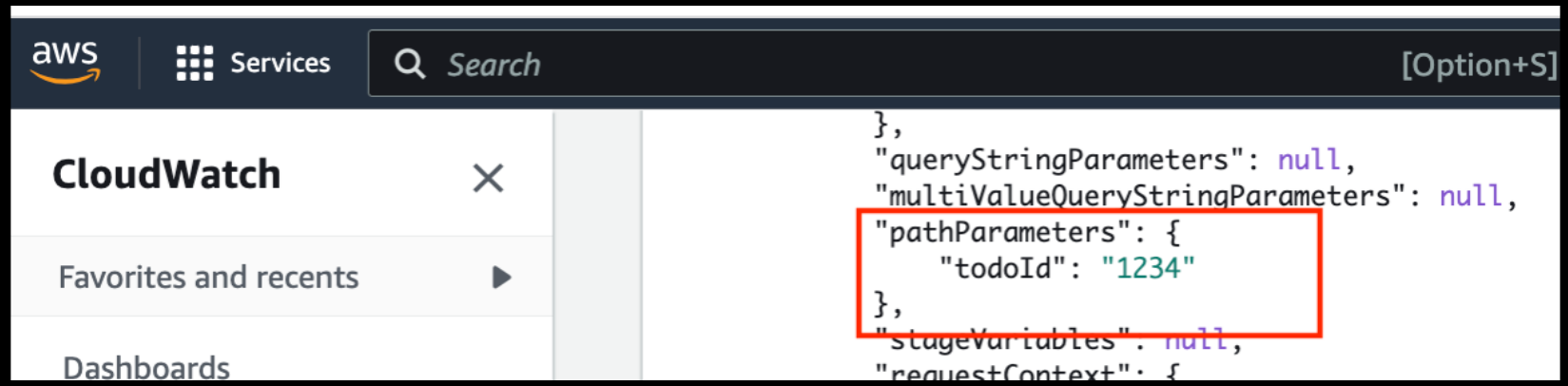
  contains=assignment&completed=false&limit=3

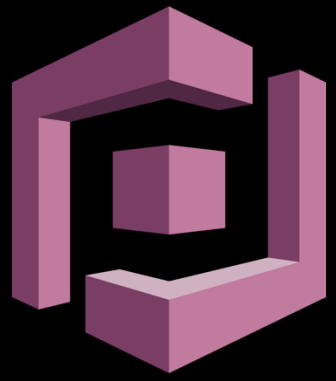


Sample

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

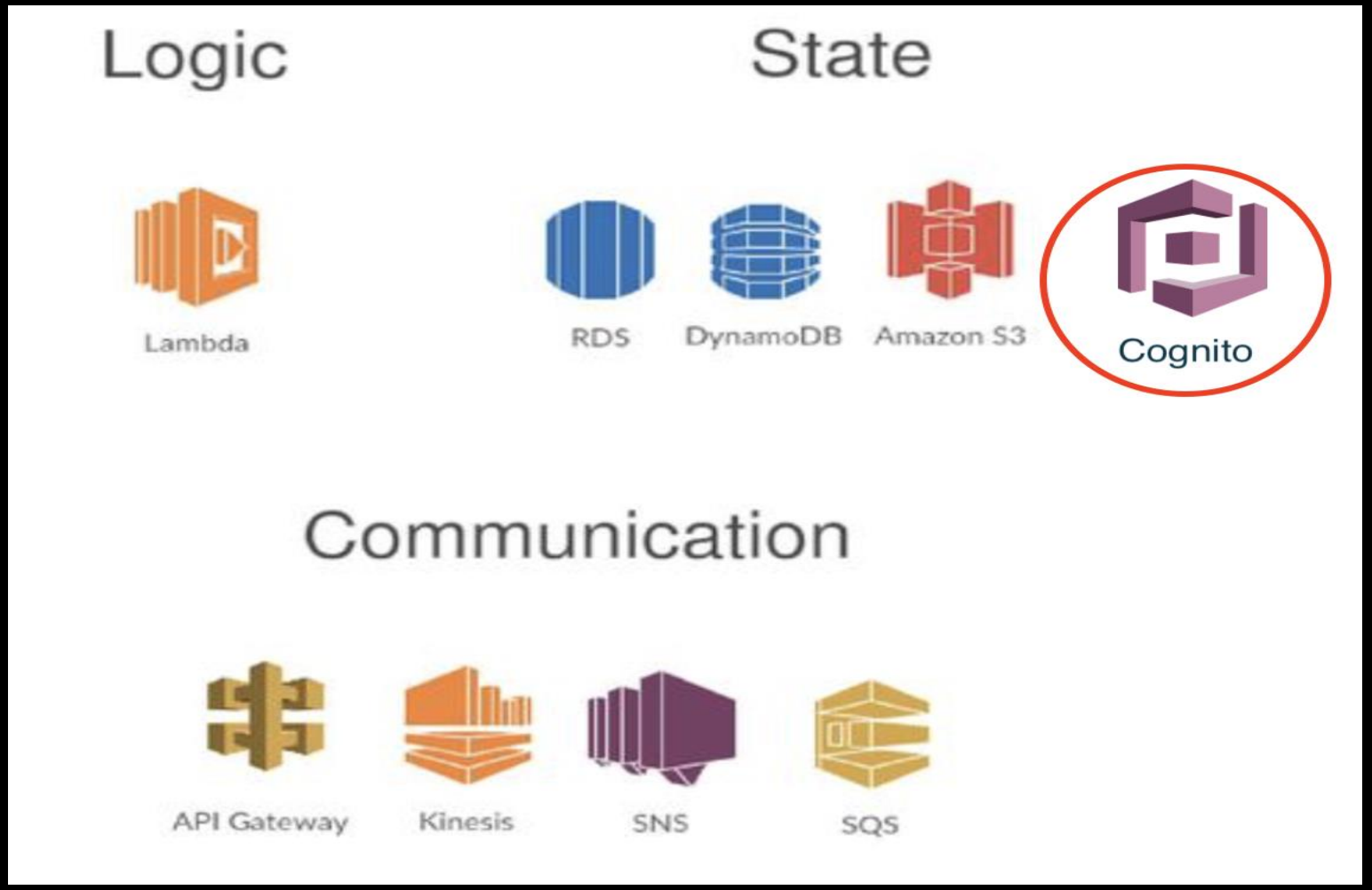
  https://ganf19s9t8.execute-api.eu-west-1.amazonaws.com/dev/todos/1234





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.

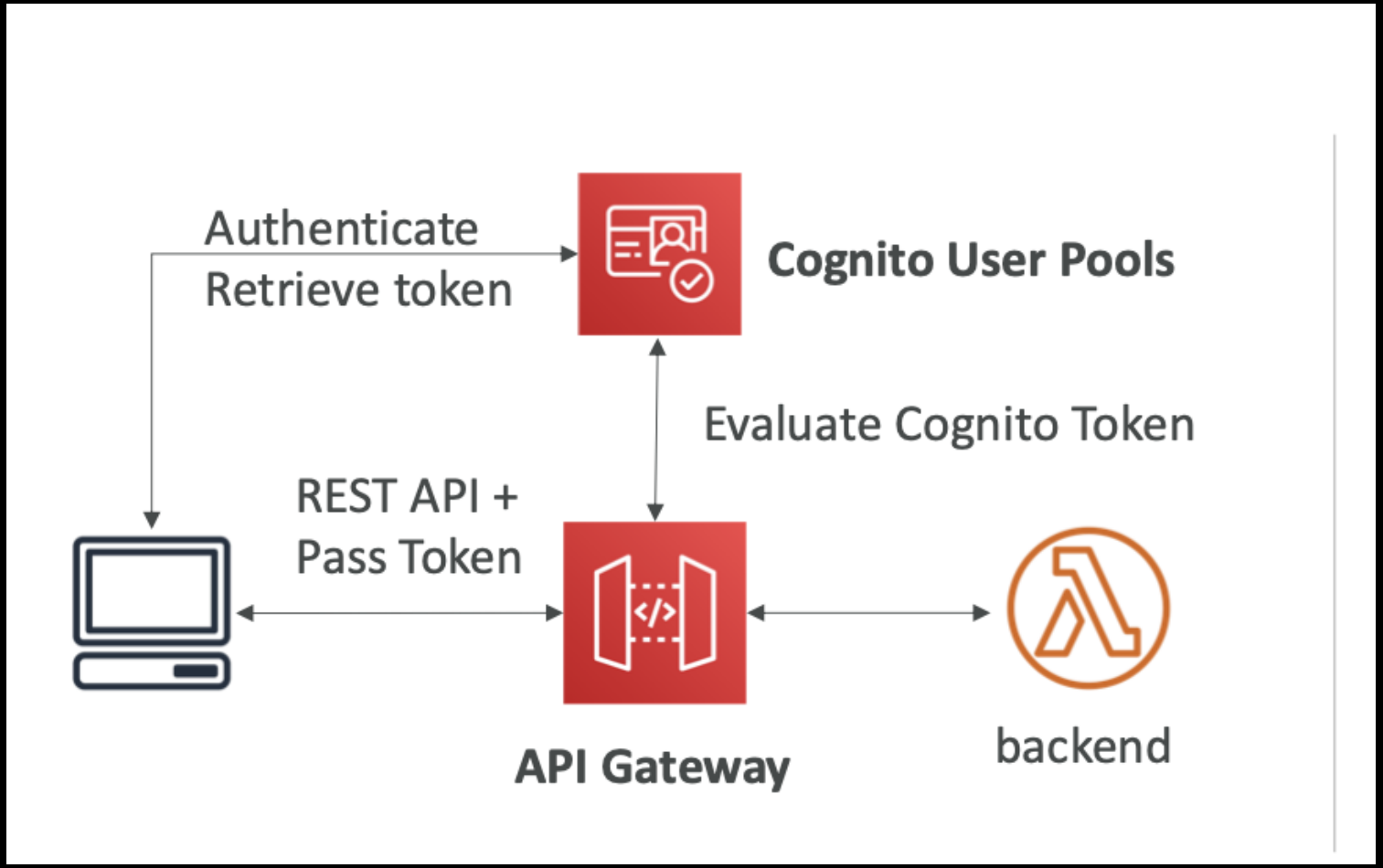
- Cognito vs IAM: “hundreds of users”, ”mobile users”.

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.

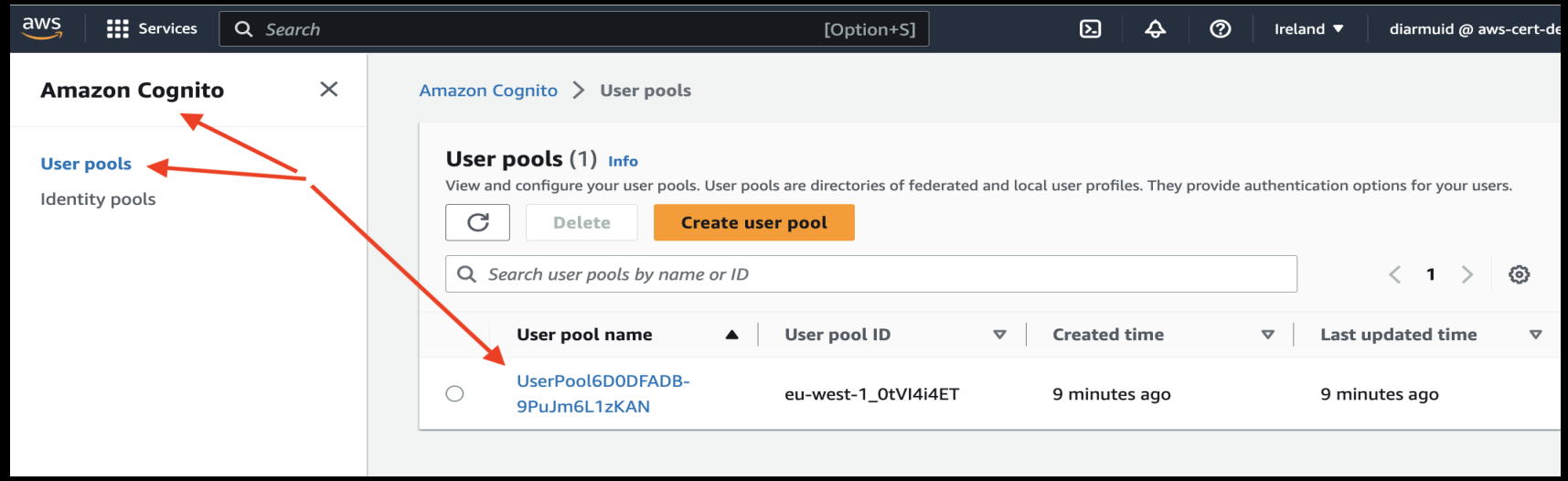
- Include JSON Web Token (JWT) in Login response.

Cognito User Pools

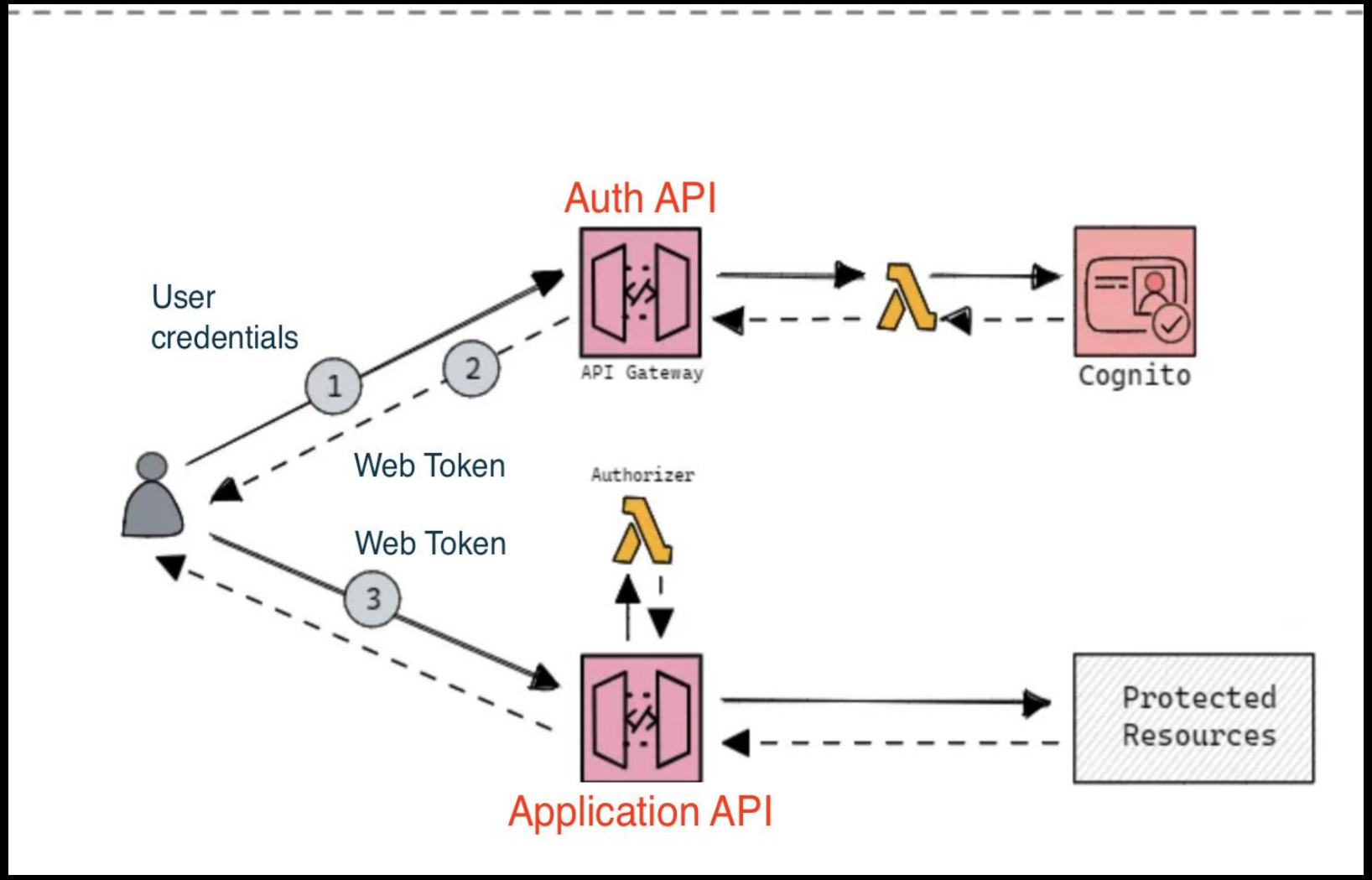


CDK code -

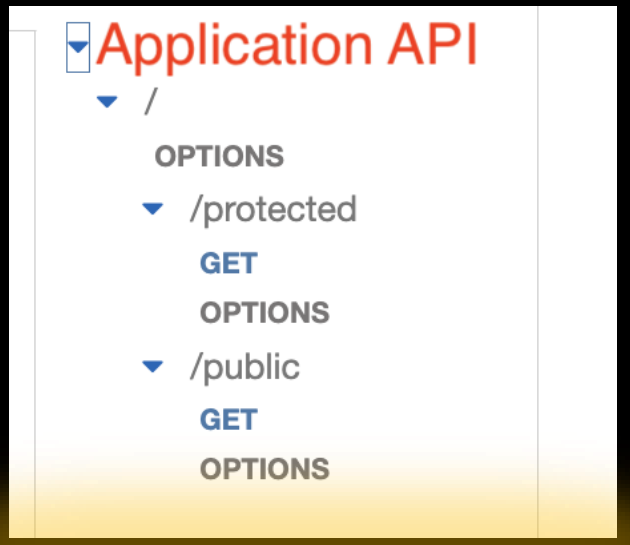
```
110  // CDK setup
111  const userPool = new UserPool(this, 'UserPool', {
112    signInAliases: { username: true, email: true },
113    selfSignUpEnabled: true,
114    removalPolicy: RemovalPolicy.DESTROY,
115  });
116
117  const appClient = userPool.addClient('AppClient', {
118    authFlows: { userPassword: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

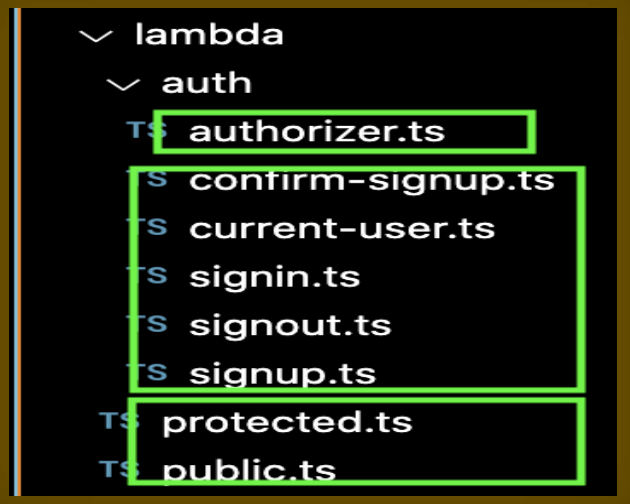


Example – Common Infrastructure setup

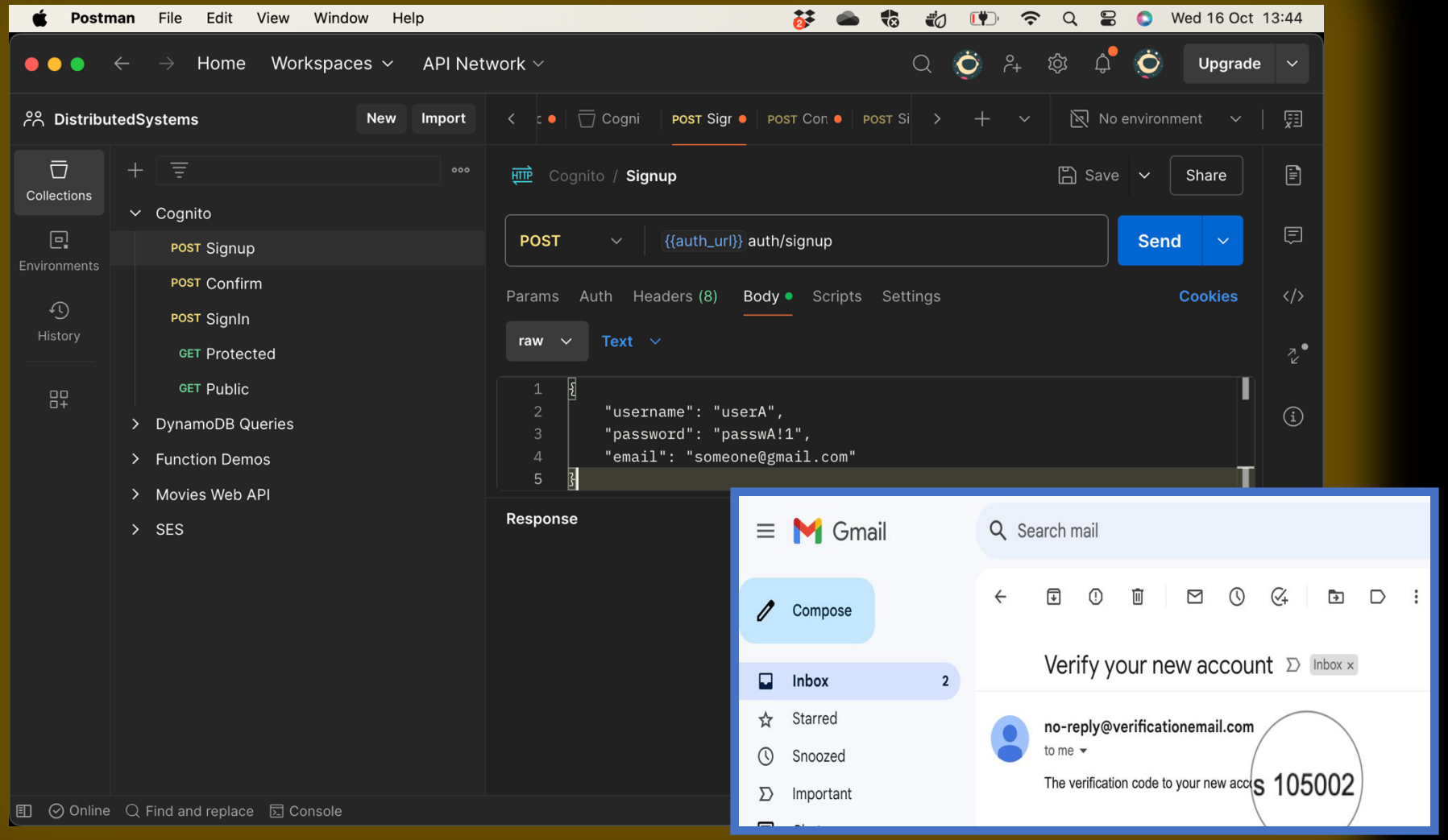


Example – The APIs





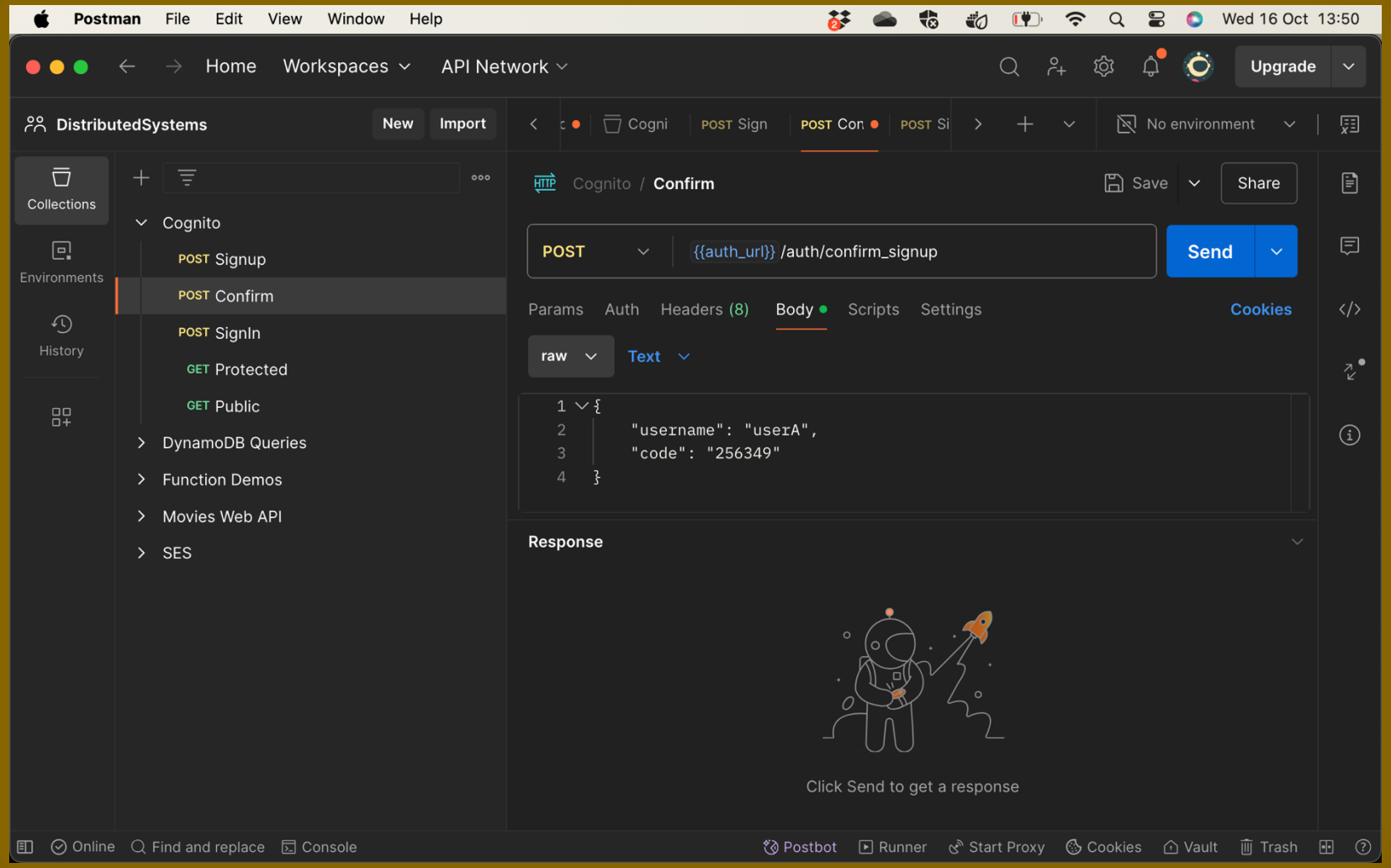
Example – Signup



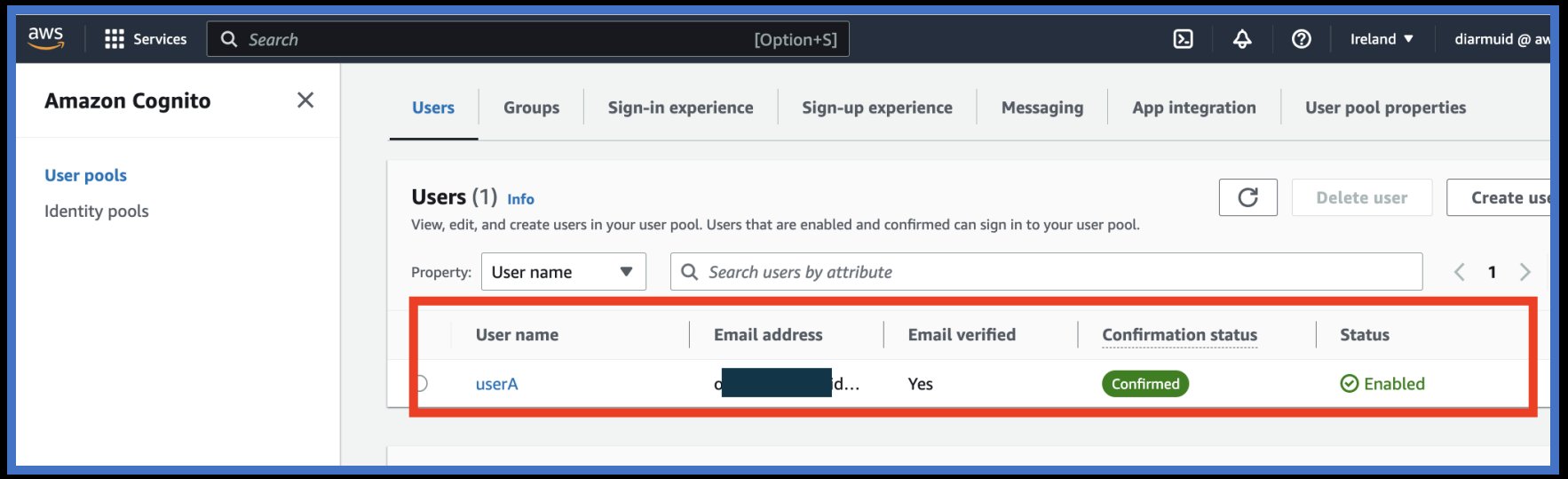
Example – Signup lambda

```
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



Example – Confirm Signup (Cognito User pool)

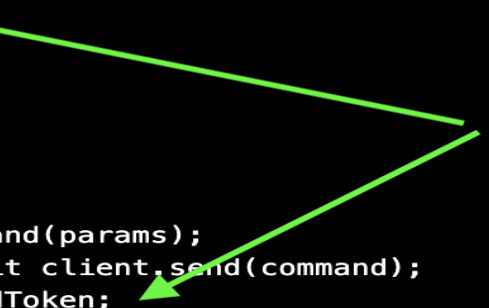


Example – Confirm Signup lambda

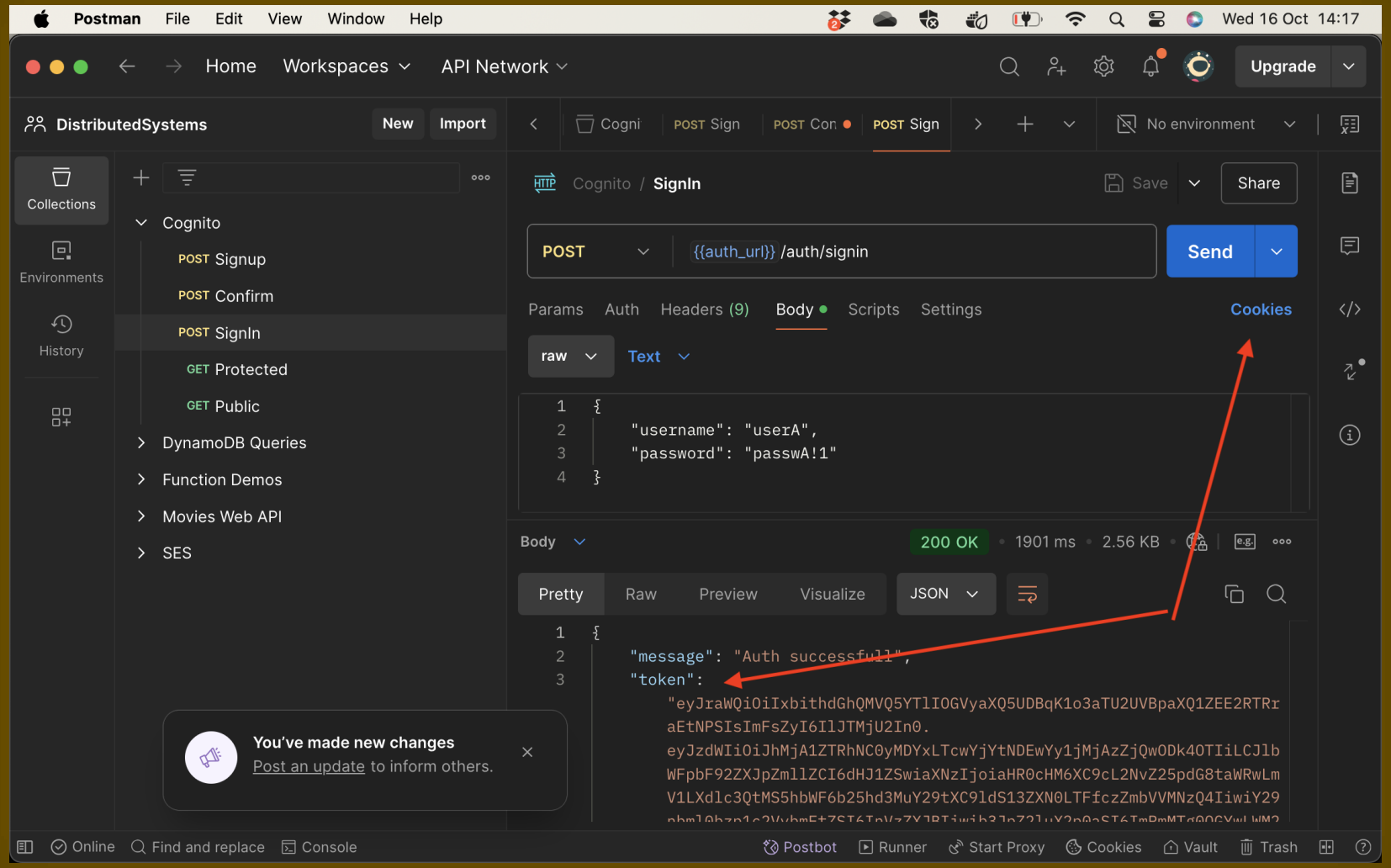
```
157  const client = new CognitoIdentityProviderClient({ region: "eu-west-1" });
158
159  type eventBody = { username: string; code: string };
160
161  export const handler: APIGatewayProxyHandlerV2 = async (event) => {
162
163    const { username, code }: eventBody = JSON.parse(event.body);
164
165    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) { ....}      You, 1 second ago • Uncommitted changes
183  };
```

Example – Sign in lambda

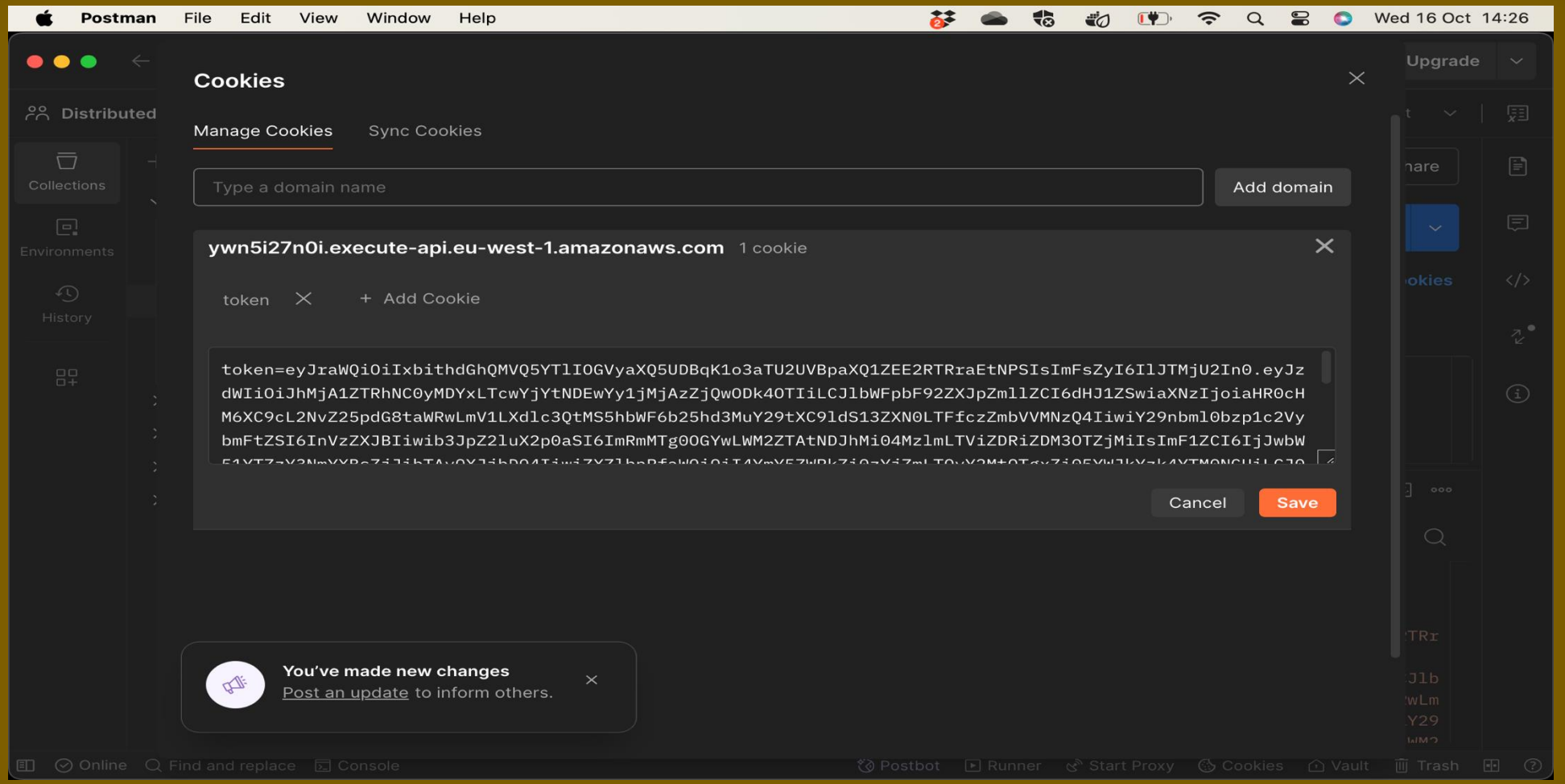
```
186
187  const client = new CognitoIdentityProviderClient({ region: "eu-west-1" });
188
189  export const handler: APIGatewayProxyHandlerV2 = async (event) => {
190    const { username, password } = JSON.parse(event.body);
191    const params: InitiateAuthCommandInput = {
192      ClientId: process.env.CLIENT_ID!,
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  };
```



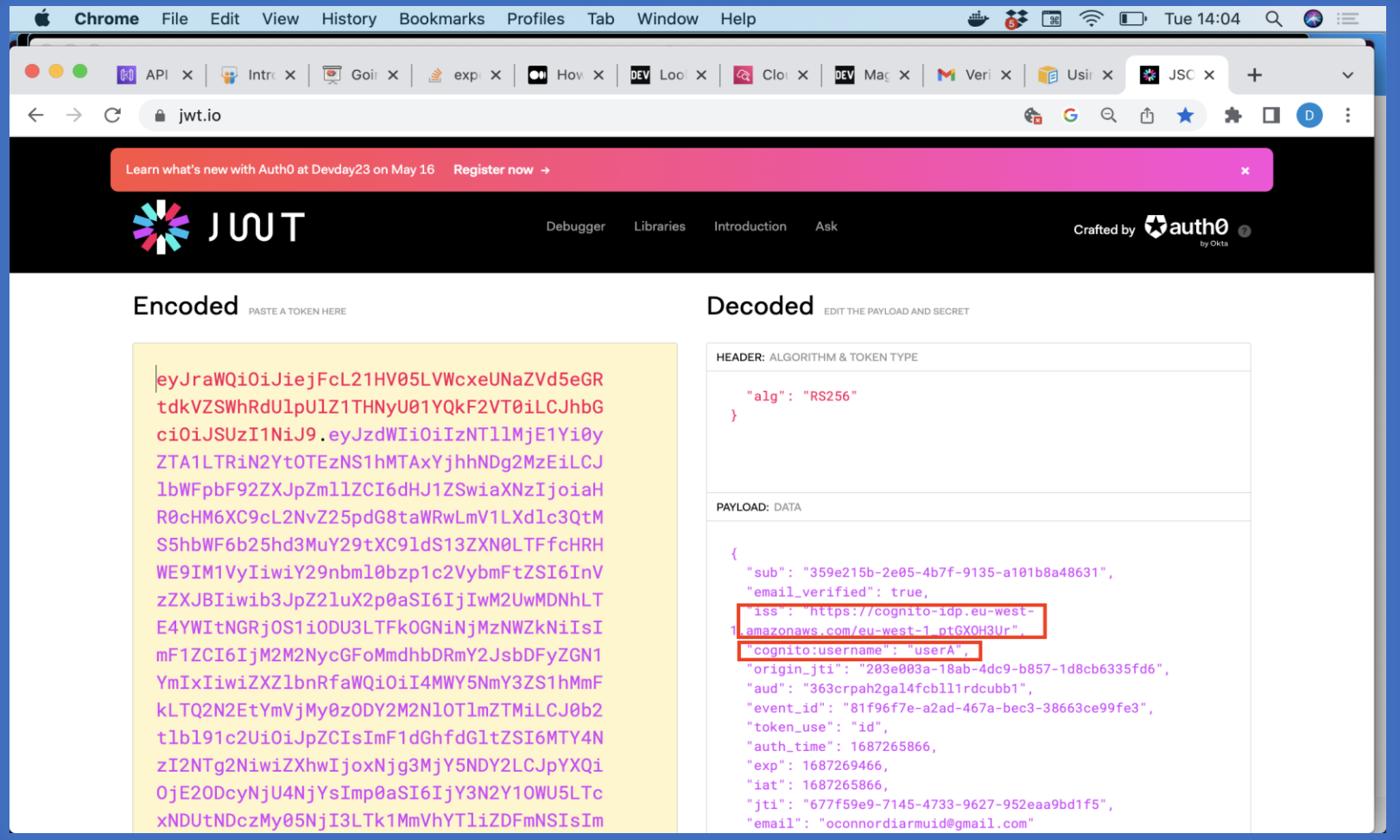
Example – Sign in request/response



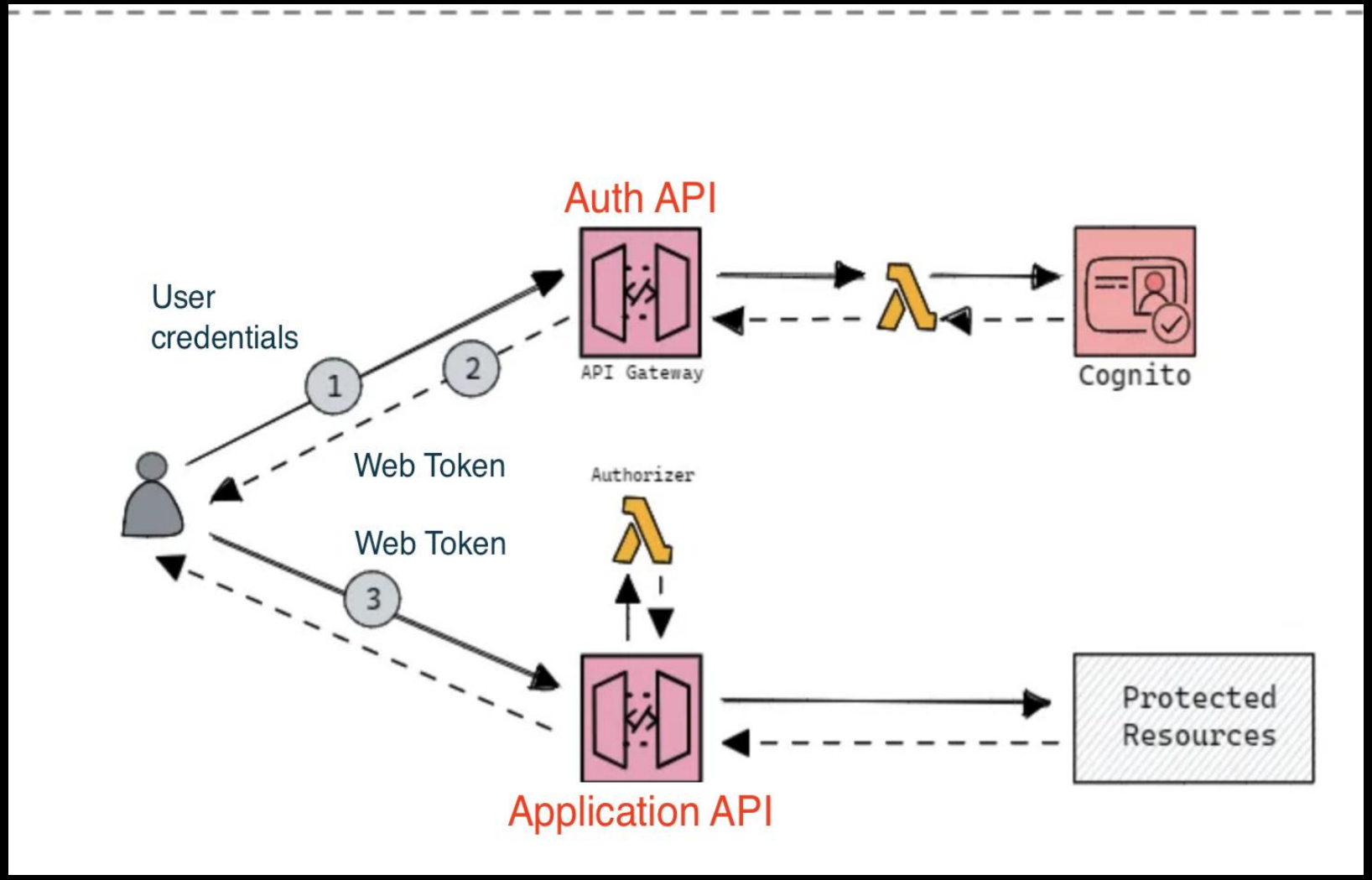
Example – Postman Cookie Management



Example – Decoded JWT token



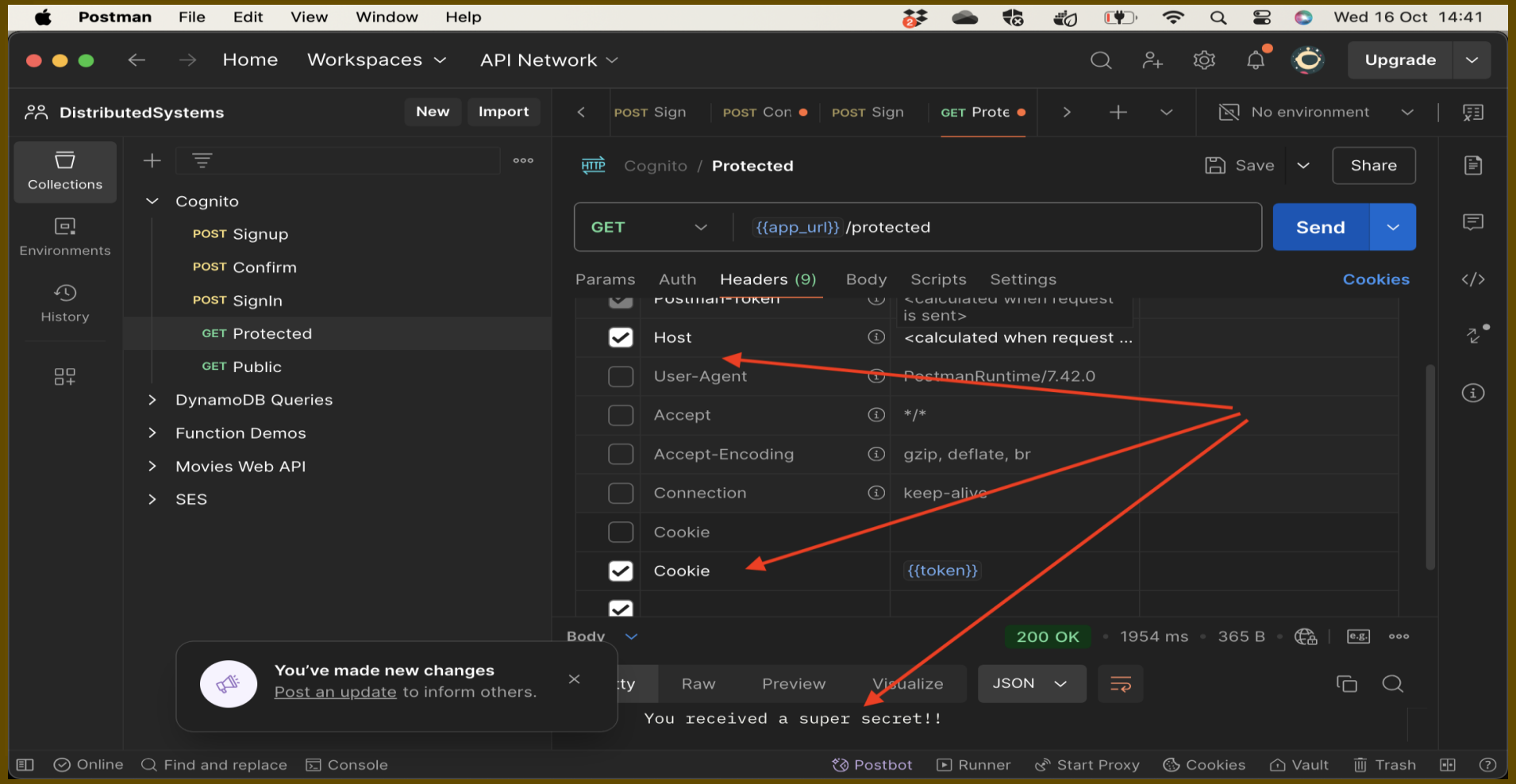
Example – Common Infrastructure setup



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));
252
```

Example – App API Protected endpoint request



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.