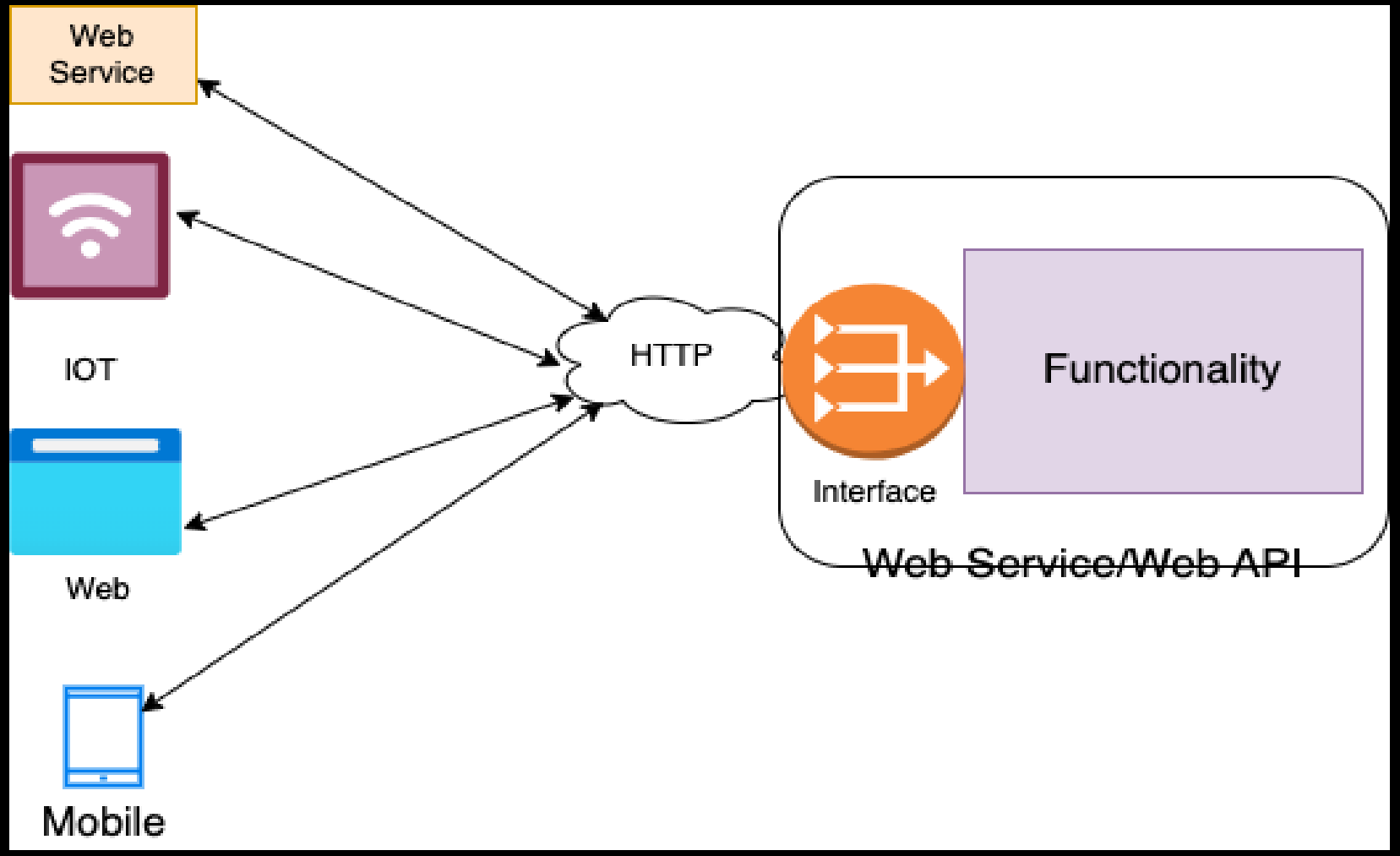
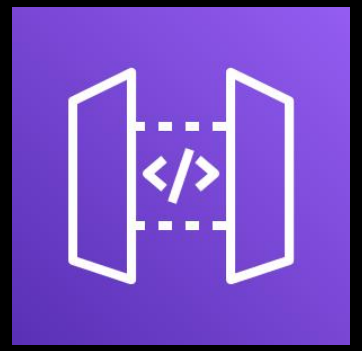
Serverless Web APIs (Contd.)

Web APIs

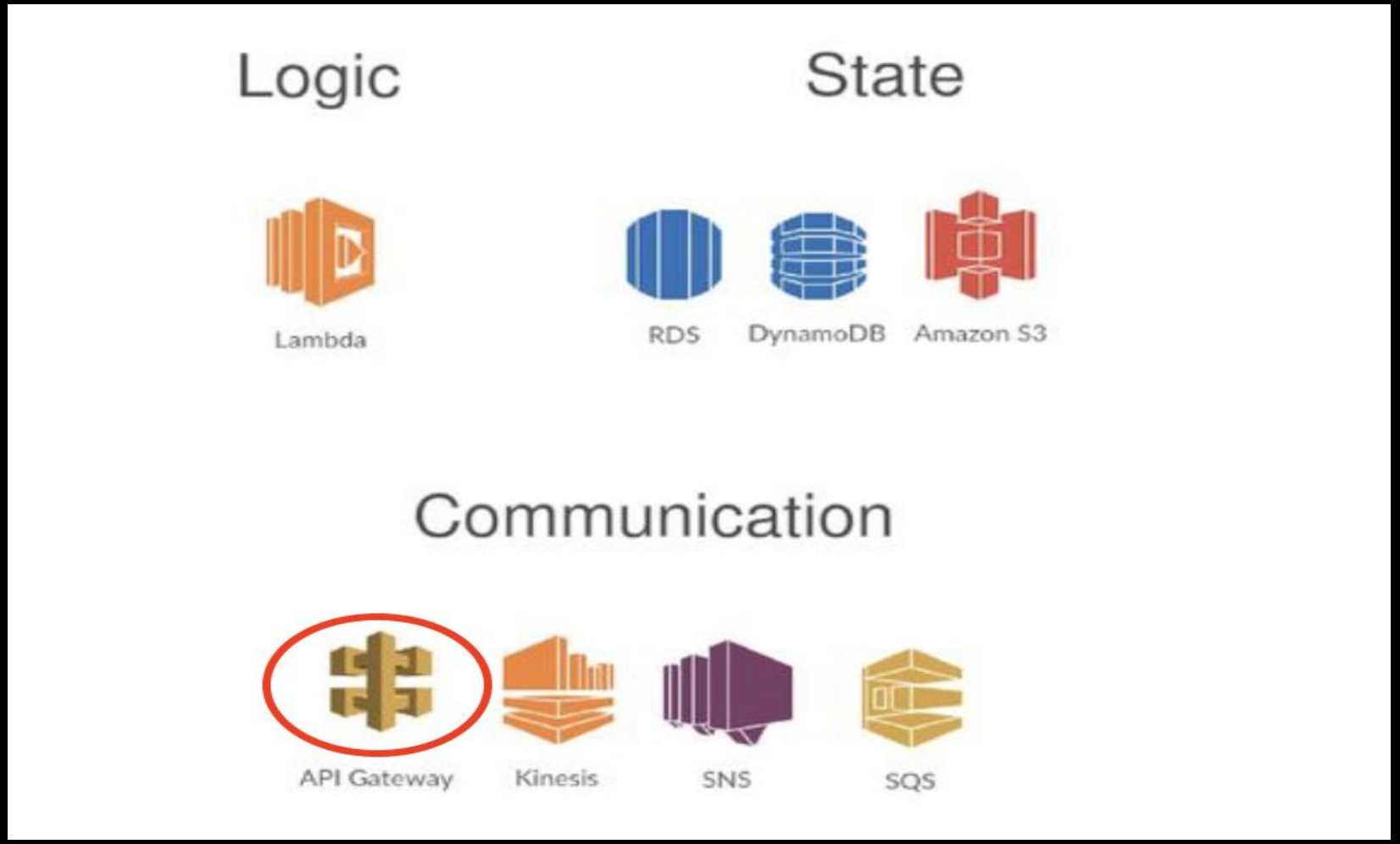

Web Service
IOT
Web
Mobile
HTTP
Interface
Functionality
Web Service/Web API



API Gateway

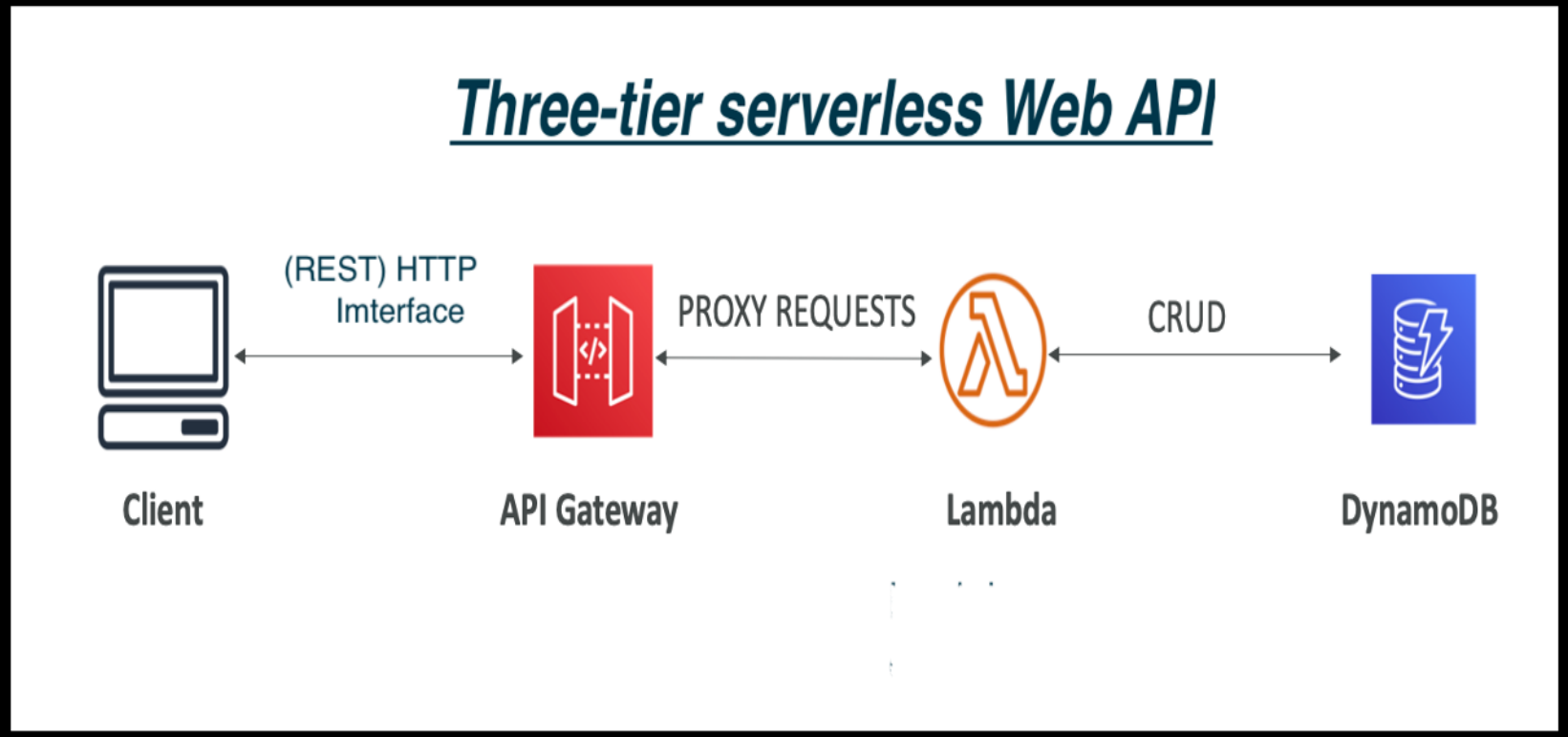
HTTP & REST Web APIs

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.



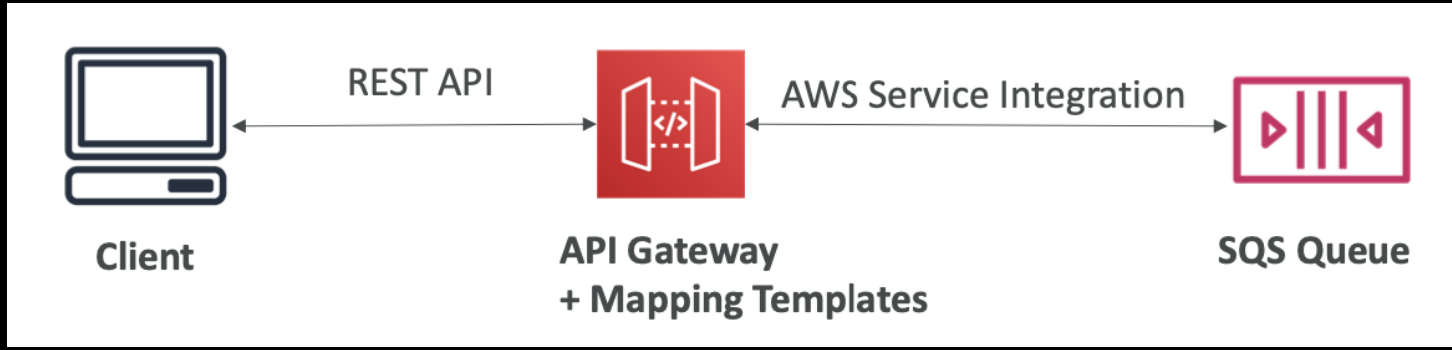
Features

- 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.

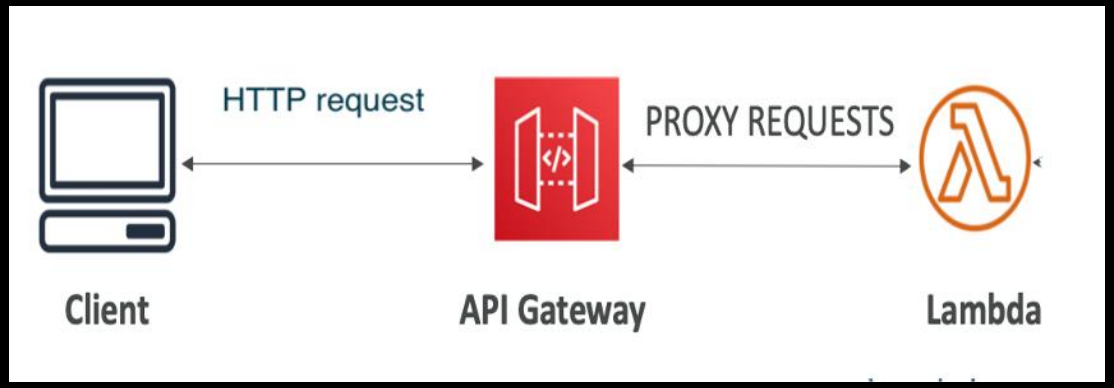
- MOCK Integration Type:
  - API Gateway returns a response without sending the request to the backend.

- 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.



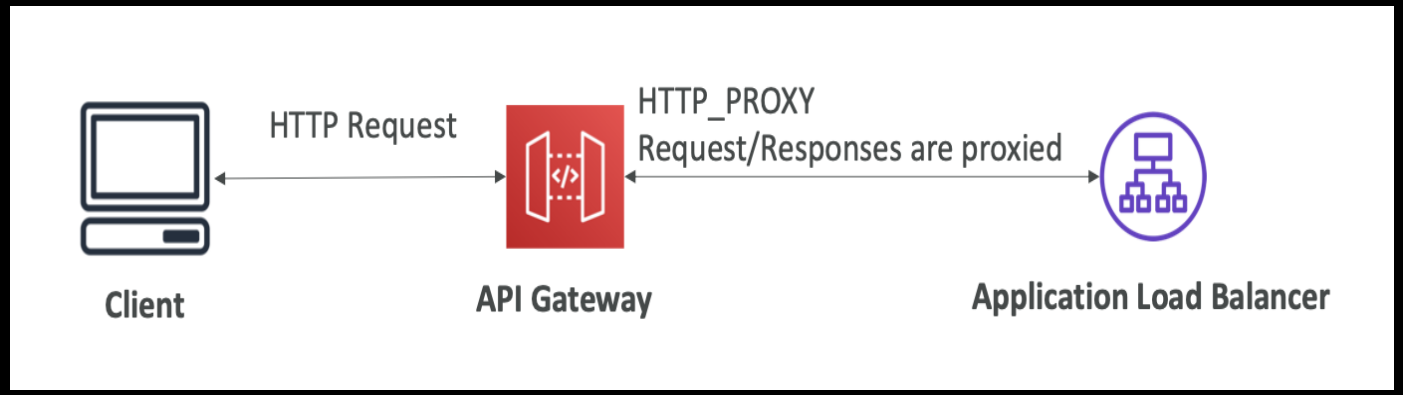
Integration Types.

- 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.



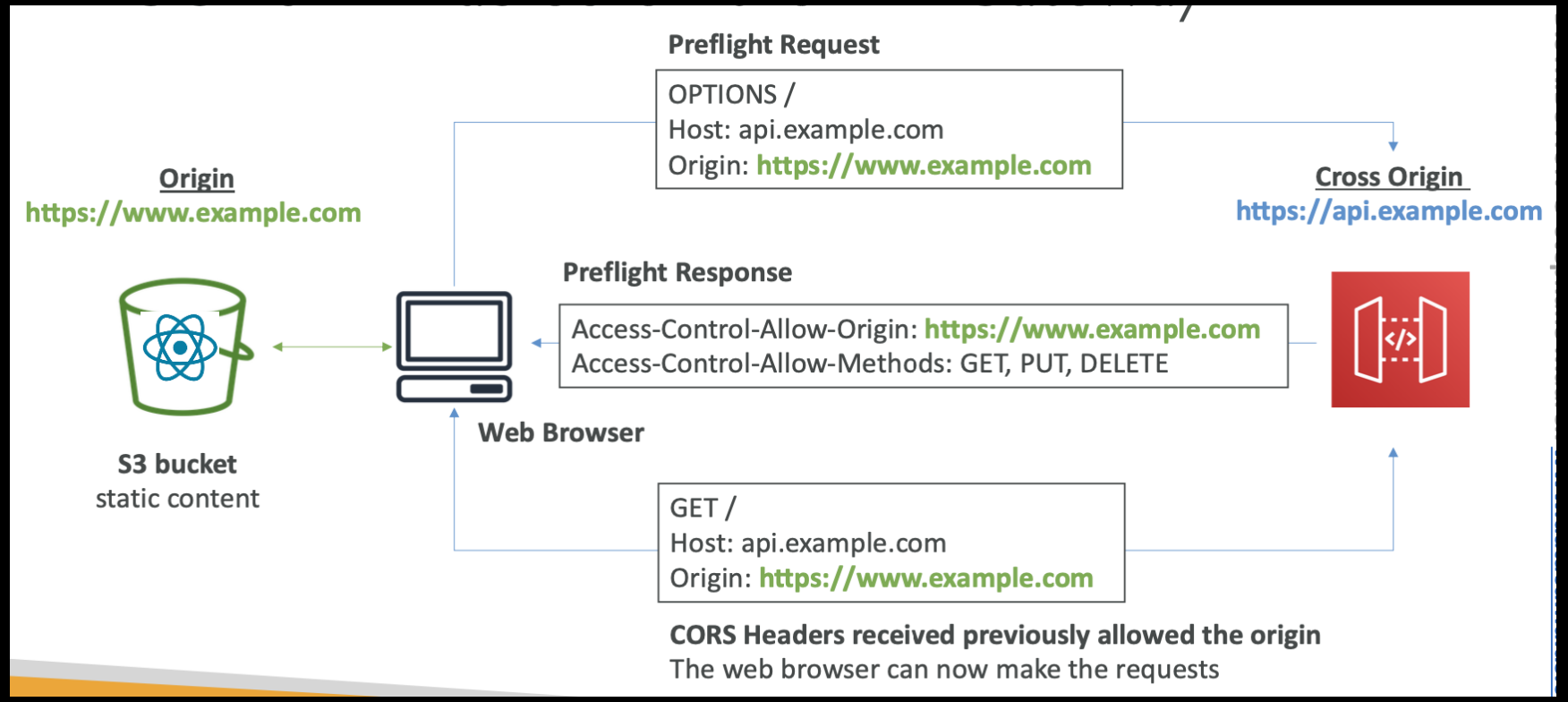
Integration Types.

- 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)

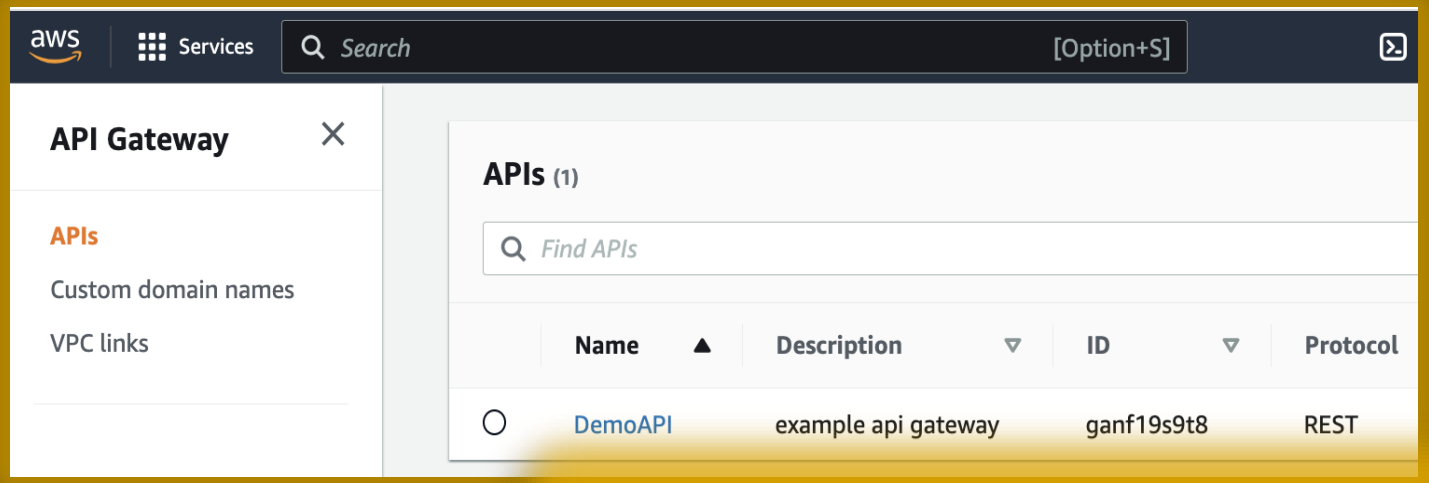
- CORS must be enabled when you receive API calls from another domain.

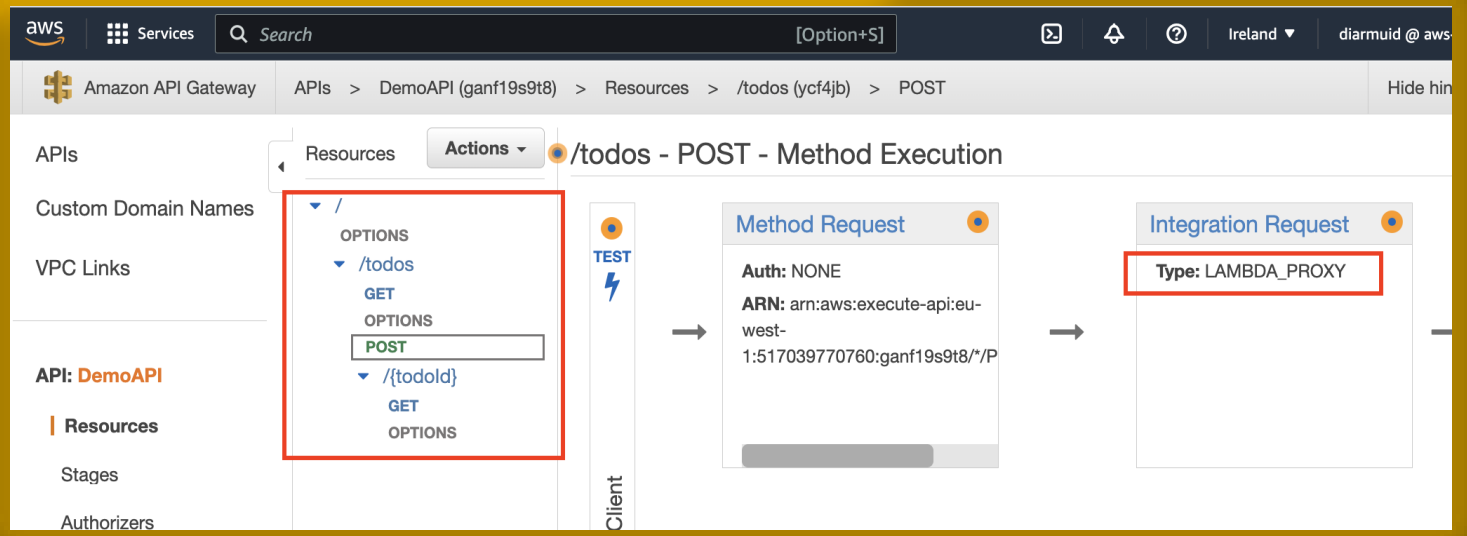


Sample

- 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 })
);
```

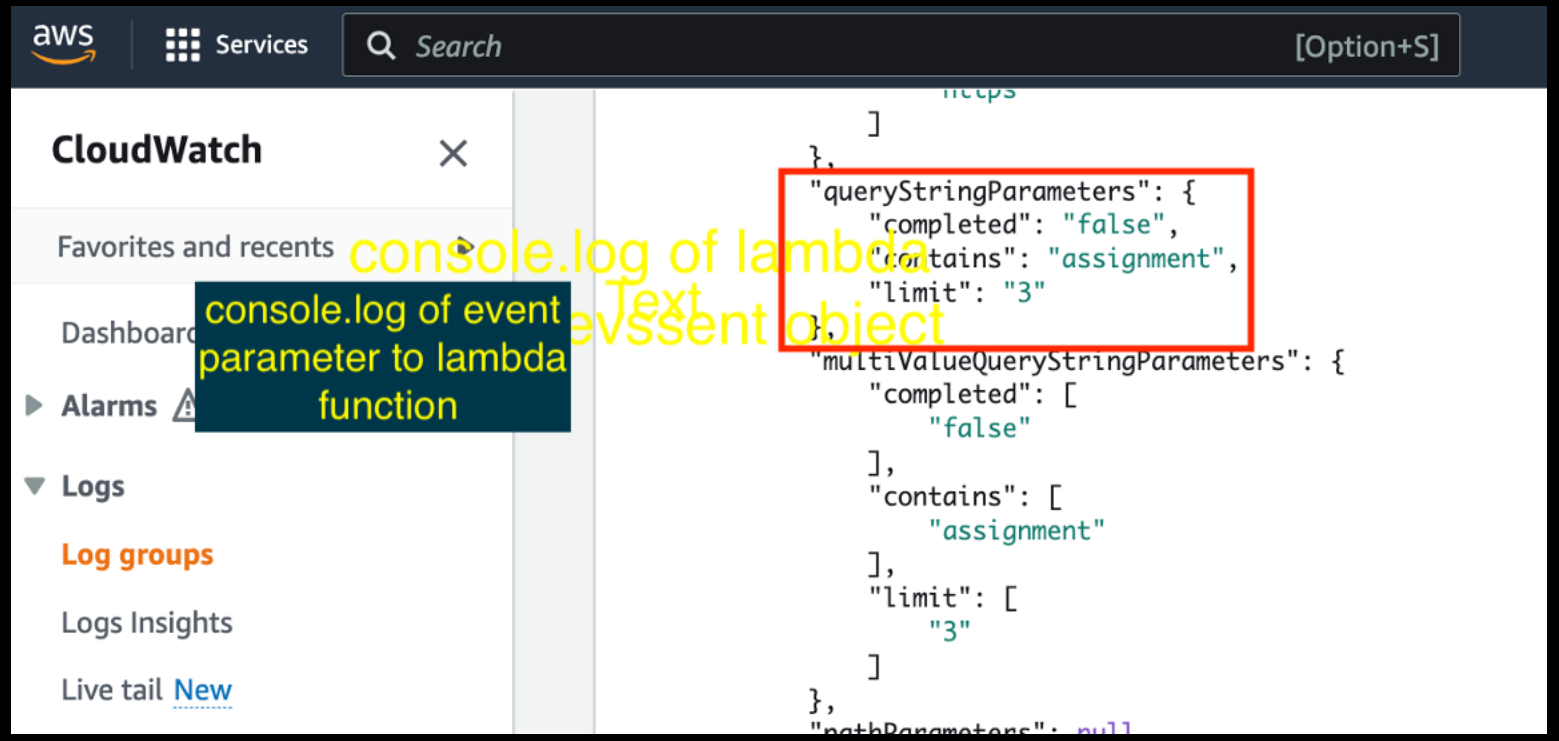
CDK stack to create REST Web API, using API Gateway service

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?

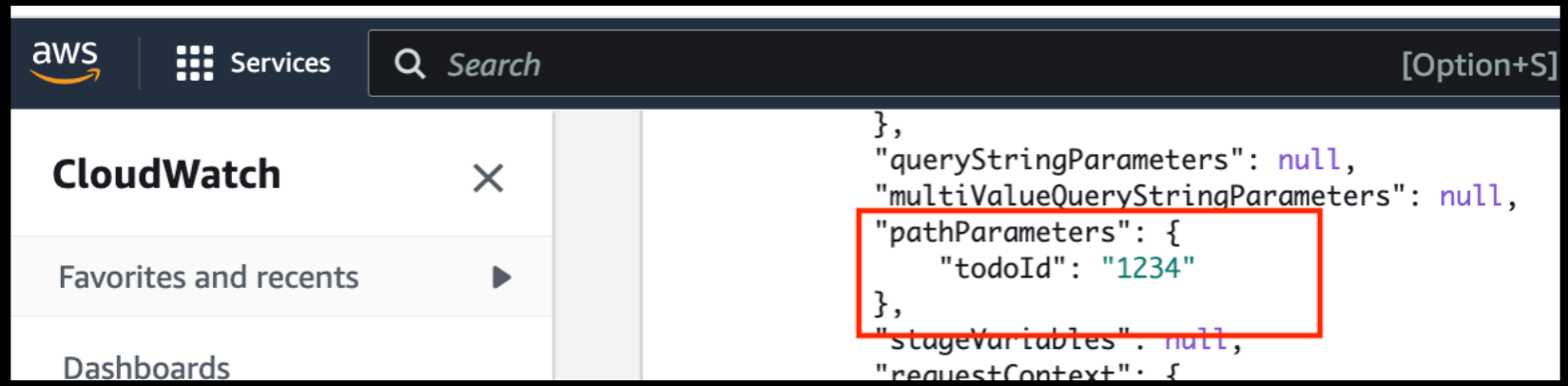
  contains=assignment&completed=false&limit=3



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



API Keys & Usage plans

- API keys are alphanumeric string values that you distribute to application developer customers to grant access to your API.
  - You can generate an API key in API Gateway, or import it into API Gateway from an external source.

- A usage plan specifies who can access one or more deployed API stages (e.g. dev, prod) and methods—and optionally sets the target request rate to start throttling requests.
  - The plan uses API keys to identify API clients and who can access the associated API stages for each key.

API Keys & Usage plans