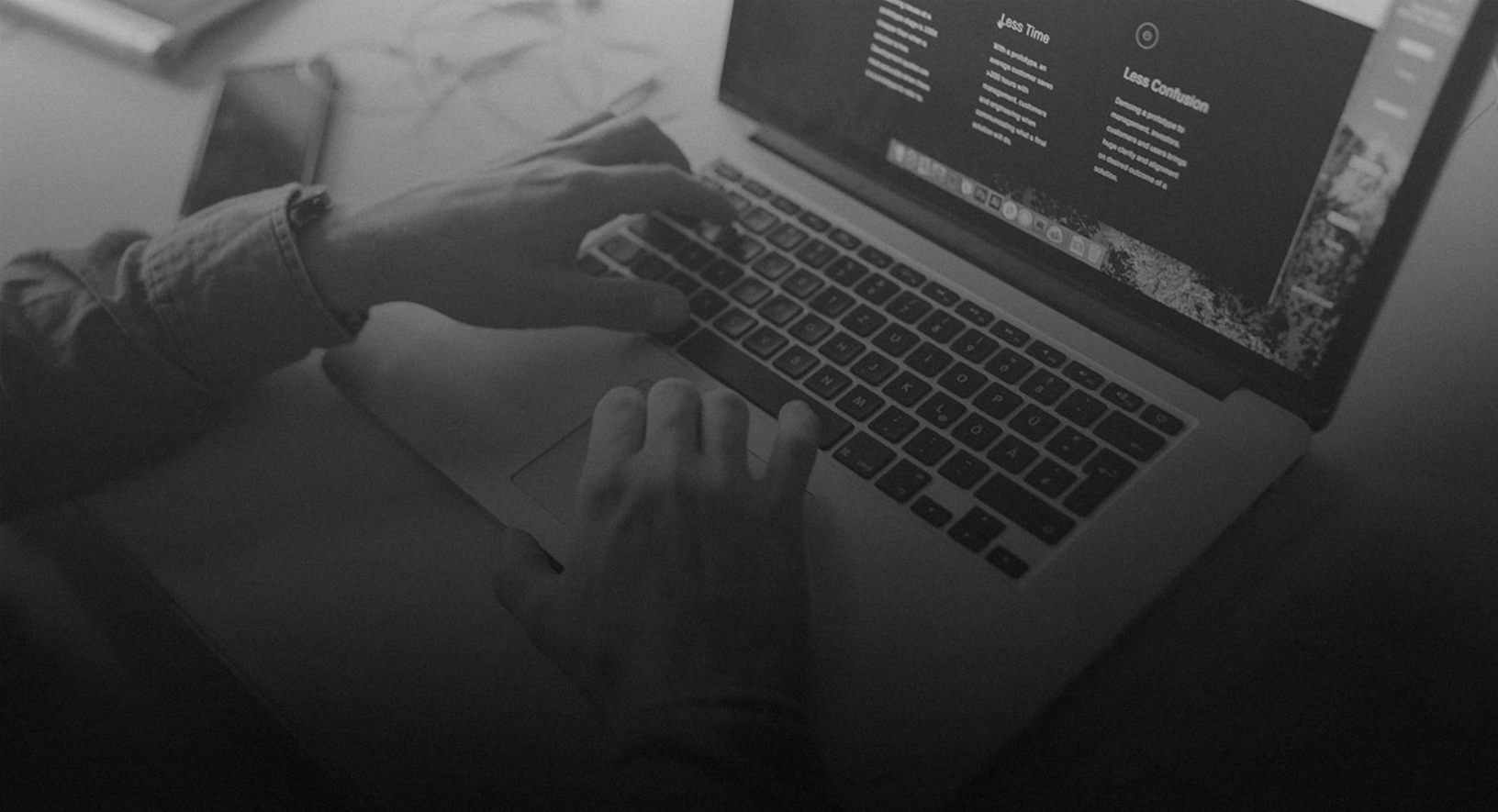


Why AWS + Auth0

Auth0

Index

Say you've polished the core parts of your app — the hardworking pieces most critical to your business. Now that you're into the late stages, you're ready to tackle the login process.

And you're realizing that identity is far more complex and challenging than expected.

Depending on your use case, choosing the right Identity Access Management (IAM)/ Customer Identity Access (CIAM) option in the AWS ecosystem is an important factor. There's a full suite of options from Cognito to Auth0, which one best suits your needs?

Identity is tough, but it doesn't have to be.

This is why Auth0 and AWS work together to ensure that AWS customers can have the quick, easily implemented identity solution they need for their specific application.

Ready to have your identity wired and running in minutes or at most days (for most common use cases)?

Check out Auth0's CEO & Co-founder Eugenio Pace explaining why Auth0 can do that for you in this interview with AWS Startup (5:29 secs) — and why we also built in deep extensibility in anticipation of your last-mile customization needs.

We work closely with AWS technical and field teams to understand and propose the right identity solution for you.. (More on why our platform is optimized for the AWS Cloud from Senior Director of Security Duncan Godfrey later in this ebook.) And 80% of Auth0's customers run on AWS.

As Auth0 CTO & Co-founder Matias Woloski explains, "AWS is the most mature cloud platform in the market. They are 5 years ahead of the rest, which means we can count on them for our scalability and reliability requirements."

From a practical standpoint, AWS Marketplace customers who choose Auth0 also get a few more benefits that make it easier to do business, such as:

- ✓ Enterprise customers can use existing AWS credits to purchase Auth0
- ✓ One invoice for AWS/Auth0
- ✓ Local currency transactions

Keep reading for more technical benefits of Auth0 on AWS, including detailed instructions on how to create AWS API Gateways with our platform and key resources.

Architected with Security in Mind

How being our own AWS use case benefits our customersAWS has built an infrastructure to "meet the requirements of the most security-sensitive organizations." It's a company, according to their CEO Andy Jassy where security is "job zero". And, it's this security culture and drive that is critical to underpinning our own.

They operate the cloud on the shared security responsibility model, meaning they are responsible for the security of the Cloud and we are responsible for the security in the Cloud.

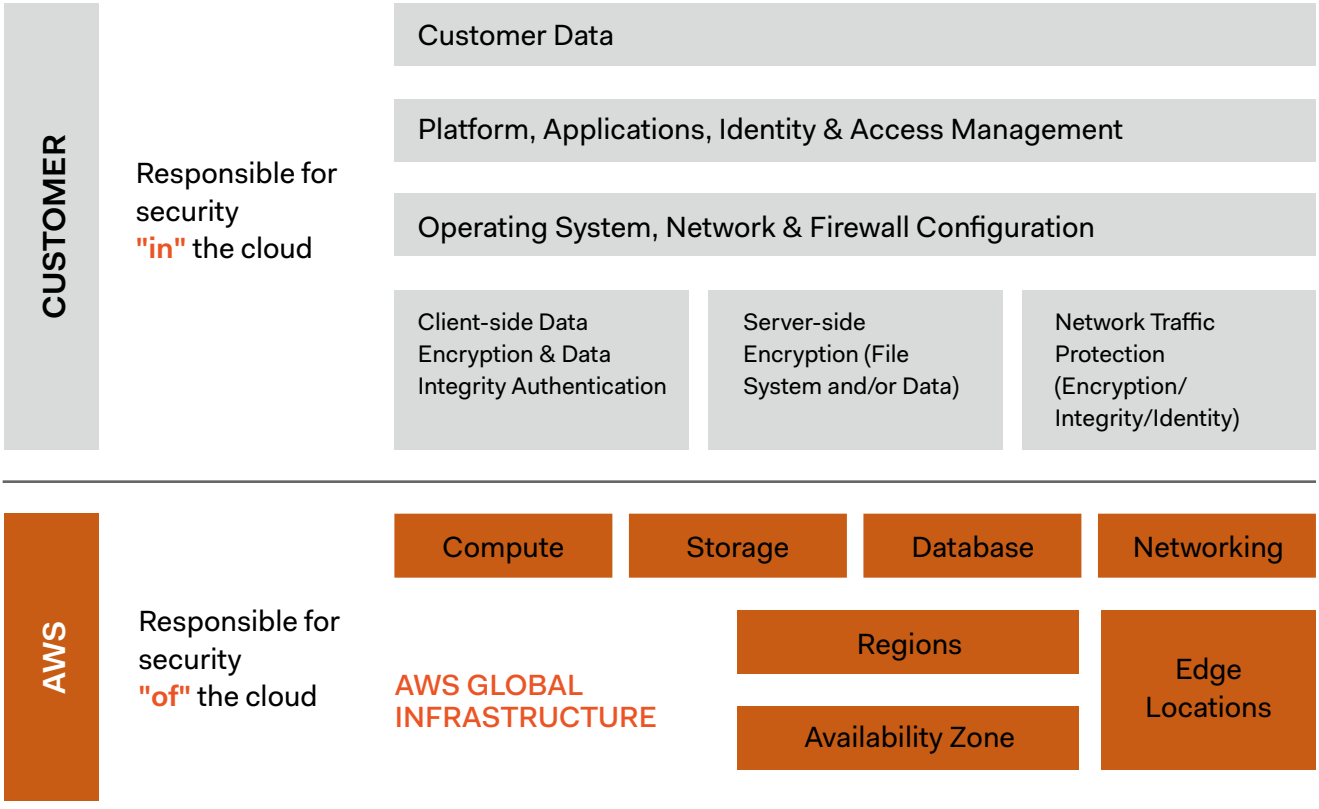


Figure 1: AWS shared security responsibility model

This model has two main benefits for Auth0's Security. Firstly, we don't have to worry about the security of the underlying infrastructure. Instead, we can focus all our security resources on protecting our application and our customers' data. For example, AWS builds some of the most secure and available data centers in the world. They specialize at building them. So by hosting our data with AWS we can give our customers confidence that their data is protected by state of the art security controls available.

Secondly, Auth0 also benefits from the requirements of all of AWS's customers. They secure the Cloud to the standards required by the federal government, NASA and the NASDAQ. The companies drive and improve AWS's security and everyone in the ecosystem benefits.

The services that AWS operated on our behalf are backed by a broad compliance regime, including: ISO 27001, PCI DSS, CSA STAR and FedRAMP.

Advanced Security Architecture — Built-in

Beyond this foundation, AWS has a number of cloud-scale security services built into its platform that simplify building and operating an advanced security architecture, including:

- ✓ Fine-grained and ephemeral access control through IAM roles
- ✓ Encryption services (including key management and data encryption)
- ✓ Detailed audit logging,
- ✓ Cloud scale network defense services with Guard Duty and AWS Shield

Using IAM has enabled us to enforce the principle of least privilege, transparently restricting our engineers to only the resources they require to perform their role. Users receive short-lived access to

perform the approved action and refresh their access periodically allowing us to move away from static long-lived credentials.

The encryption services that AWS provides allow us to implement critical encryption controls at scale with low friction for Security and engineering teams. The AWS Key Management Services (KMS) is a fips-140 validated key store that has allows us to protect and rotate our keys. And, services such as The Elastic Block Store (EBS) encryption allows us to encrypt all our data at rest by default without managing our own infrastructure.

The AWS audit service CloudTrail provides us with the detailed granular audit logging for every change in our environment. AWS is an API driven infrastructure and every call to those APIs creates an audit log, thus providing the security team with deep visibility into our environment. These can be used to build detections for malicious activities and to perform an investigation into possible incidents, by providing a detailed picture of what happened.

AWS has also built network defense services that allow us to protect our cloud-scale service. Being behind the protections of AWS Shield means our service receives the same protections as the biggest infrastructure helping to protect our availability. We also use GuardDuty to continuously monitor for malicious activity and unauthorized behavior in our AWS accounts and workloads.

Roadmap input, early emerging threat detection

And finally, AWS is our security partner. By using AWS we are part of a wider ecosystem of security teams working for the same goal. We work closing closely with our account team to understand both emerging threats and their development roadmaps. We also meet regularly to discuss issues, hear about new developments, and suggest new features.

By using AWS we believe we are using the best platform for building security available. We make use of this to build a world-class security service that is required to support an identity platform.

AWS Increases Security Scan Freedom

Thanks to a recent change to the AWS penetration testing policy, AWS customers now have more freedom when performing security scans against their AWS resources. At Auth0 we're leveraging this policy change to better understand and protect the cloud resources we expose to the world.

Before the policy was updated, AWS customers were required to submit a penetration testing authorization request and wait for approval before being permitted to perform a scan. This policy change allows penetration testing of many AWS services without the need for pre-approval.

What This New Policy Means for Security Teams

Almost every cloud security engineer has had the experience of filling out an AWS penetration testing request and waiting on a response. This manual process greatly reduced the velocity at which you could perform vulnerability scans. For organizations that push code to production often this left security teams playing a constant game of "catch-up".

Or maybe you've been the security engineer tasked with performing an external vulnerability scan of your AWS environment and were left guessing what Internet-facing resources even exist. Cloud providers like AWS make it so easy to put services on the Internet making it even harder to control what ports and protocols you're exposing to the world.

Daniel Miessler said it well in his recent blog post on this topic, ./getawspublicips.sh: Know the Public AWS IPs You Have Facing the Internet:

> *"The biggest problem I see however — by a wide margin — is companies not having any idea what ports, protocols, and applications they are presenting to the world."*
>
> **Daniel Miessler**

Or you could be the security engineer receiving an alert or a vulnerability report with a public IP address that is almost impossible to track down inside your dynamic cloud environment. That IP could be long gone, released from your load balancer, and now associated with something and someone else entirely. The updated penetration testing policy opens the door to solving these problems.

What We Are Doing at Auth0

The Auth0 Cloud Security team is taking advantage of this policy change to improve the way we track and protect our Internet footprint. In the past, we've relied on scheduled external scans of our AWS environment to validate what services, ports, and protocols we were exposing. We typically were left performing these scans monthly or quarterly, simply due to the manual overhead of getting approval and scanning our entire environment within the allotted time window. Since we have over 120 AWS accounts and operate out of several regions, this was a real challenge.

The first step to enabling continuous external vulnerability scans was getting a full view of our public IP footprint. With the dynamic nature of public IP assignment to things like ELBs and CloudFront distributions, this required some engineering work. Fortunately, we already had internal tooling that continuously crawls our AWS environment to capture the public IPs being used. This tool consists of

some simple Lambda functions that store public IP data in a DynamoDB table.

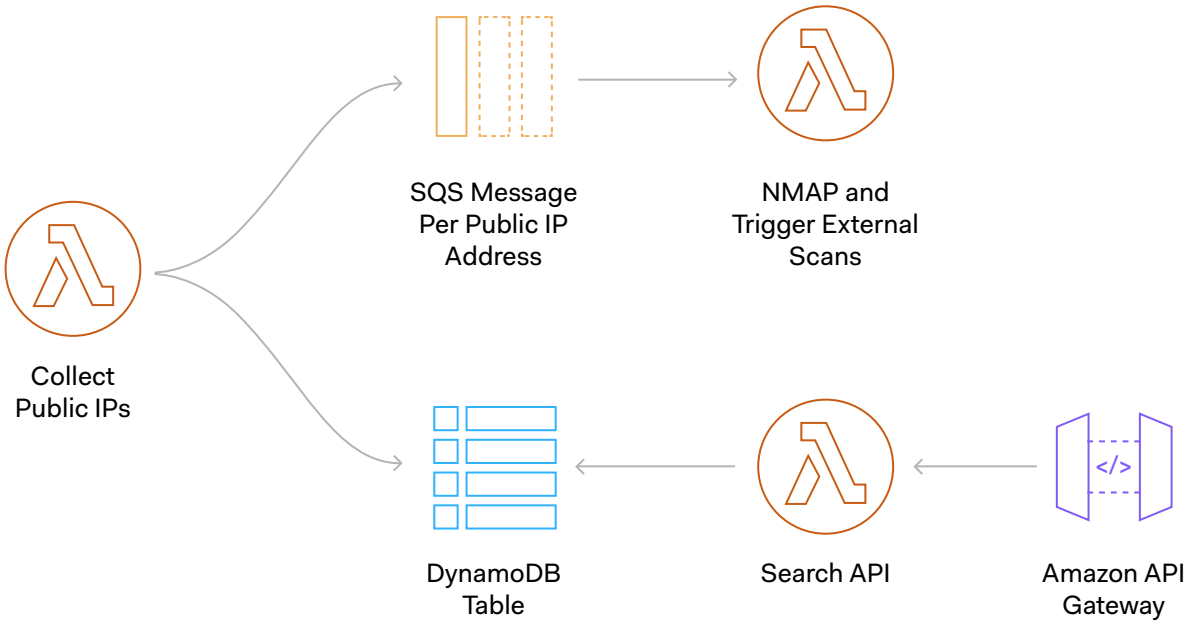
To make this data easily consumable we built a small REST API with a few query endpoints. This is a great tool to have at our fingertips when performing network forensics or investigating potential vulnerability findings that refer to public IP addresses. An example DynamoDB record is shown below.

```
{
 "ipAddress": "x.x.x.x",
 "resourceId: "xyz.us-east-1.elb.amazonaws.com",
 "resourceName": "load-balancer-01",
 "resourceType": "AWS::ElasticLoadBalancingV2::LoadBalancer",
 "account": "12345678",
 "region": "us-east-1"
 "lastDiscovered": "2019-03-14T20:00:44.160144"
}
```

Next, we identified how we could use this public IP data to continuously scan our Internet footprint for unwanted services and vulnerabilities. We added some logic to our public IP discovery Lambda to feed a Simple Queue Service (SQS) queue with each discovered IP. Another Lambda function picks up that SQS message and does a few things:

- ✓ It executes a Network Mapper (NMAP) scan against that IP address. We achieved this by crafting a Lambda Layer that contains the NMAP binary and supporting files allowing Lambda itself to execute the NMAP command.
- ✓ It then feeds the IP address into other vulnerability scanning tools we use to run longer, more in-depth assessments against services listening at that IP address.

Here is a high-level view of how the solution is put together.



We've set this up to run continuously across all the accounts and regions we utilize. Getting this outside-in view of our cloud infrastructure provides many benefits:

- ✓ We can continuously detect potential vulnerabilities in our Internet-facing services
- ✓ We can be alerted when a new Internet-facing resource is provisioned
- ✓ We can detect overly permissive security groups which expose more ports than necessary to the Internet
- ✓ We are enabled to better perform forensics knowing our public IP footprint at any point in time

Moving From Manual to Automated

The latest AWS penetration testing policy update is great news for security teams. At Auth0 we've been able to convert a once-manual and time-consuming process into an automated and continuous defensive control. After running our solution for a week we collected over 3,800 public IP addresses used by our AWS resources. We are continuing to enhance our tooling and hope to open source much of what we have built in the near future.

How to set up an AWS/API gateway

One of the things that happens when you combine AWS with Auth0 is that identity just becomes easier — even things like setting up an API gateway (if you need one). In this chapter, we are going to go over the AWS API Gateway and authenticate it all with Auth0. You will need an Auth0 account, you can sign up for free here or login to an existing account here.

Plan:

- ✓ Define what the value of an API Gateway is and what it is.
- ✓ Define what are the elements that come at play in the architecture.
- ✓ Show an example of API Gateway.
- ✓ Show a diagram of what this looks like.
- ✓ Define why you need authentication on this architecture and how to do it.
- ✓ Show an example of API Gateway with Authentication.

With AWS, you can create powerful, serverless, highly scalable APIs and applications using Lambda and API Gateway for the client.

When you use AWS Lambda, you are able to run code without having to manage your own servers. AWS Lambda will execute your code when needed and scale it automatically. It will take care of things like operating system maintenance, code monitoring, and logging.

The API Gateway extends the capabilities of Lambda by adding a service layer in front of your Lambda functions to extend security, manage input and output message transformations, and provide capabilities like throttling and auditing. A serverless approach simplifies your operational demands since concerns like scaling out and fault tolerance are now the responsibility of the compute service that is executing your code.

This tutorial will show you how to set up your API with API Gateway, create and configure your Lambda functions (including the custom authorizers) to secure your API endpoints, and implement the authorization flow so that your users can retrieve the Access Tokens needed to gain access to your API from Auth0.

More specifically, the custom authorizers will:

- ✓ Confirm that the Access Token has been passed via the **authorization** header of the request to access the API
- ✓ Verify the RS256 signature of the Access Token using a public key obtained via a JWKS endpoint
- ✓ Ensure the Access Token has the required Issuer "**iss**" and Audience "**aud**" claims

* ***NOTE**: New to OAuth2.0? Check out our introduction to OAuth 2.0.*

To that end, this tutorial will be divided into the following sections.

Step 1 - Configure Auth0

Step 2 - Set up and deploy an AWS API Gateway

Step 3 - Create Custom Authorizers

Step 4 - Secure an API Using Custom Authorizers

How API Gateway Custom Authorizers Work

According to Amazon, an API Gateway custom authorizer is a "Lambda function you provide to control access to your API using bearer token authentication strategies, such as OAuth or SAML."

Whenever someone (or some program) attempts to call your API, API Gateway checks to see if there's a custom authorizer configured for the API.

If there is a custom authorizer for the API, API Gateway calls the custom authorizer and provides the authorization token extracted from the request header received.

You can use the custom authorizer to implement different types of authorization strategies, including JWT verification, to return IAM policies authorizing the request. If the policy returned is invalid or if the permissions are denied, the API call fails.

For a valid policy, API caches the returned policy, associating it with the incoming token and using it for the current and subsequent requests. You can configure the amount of time for which the policy is cached. The default value is 300seconds, and the maximum length of caching is 3600 seconds (you can also set the value to 0 to disable caching).

Before You Begin

Before beginning this tutorial, you'll need to sign up for an AWS account. This grants you access to all of the AWS features we'll use in this tutorial, including API Gateway and Lambda. All new members receive twelve months of free tier access to AWS.

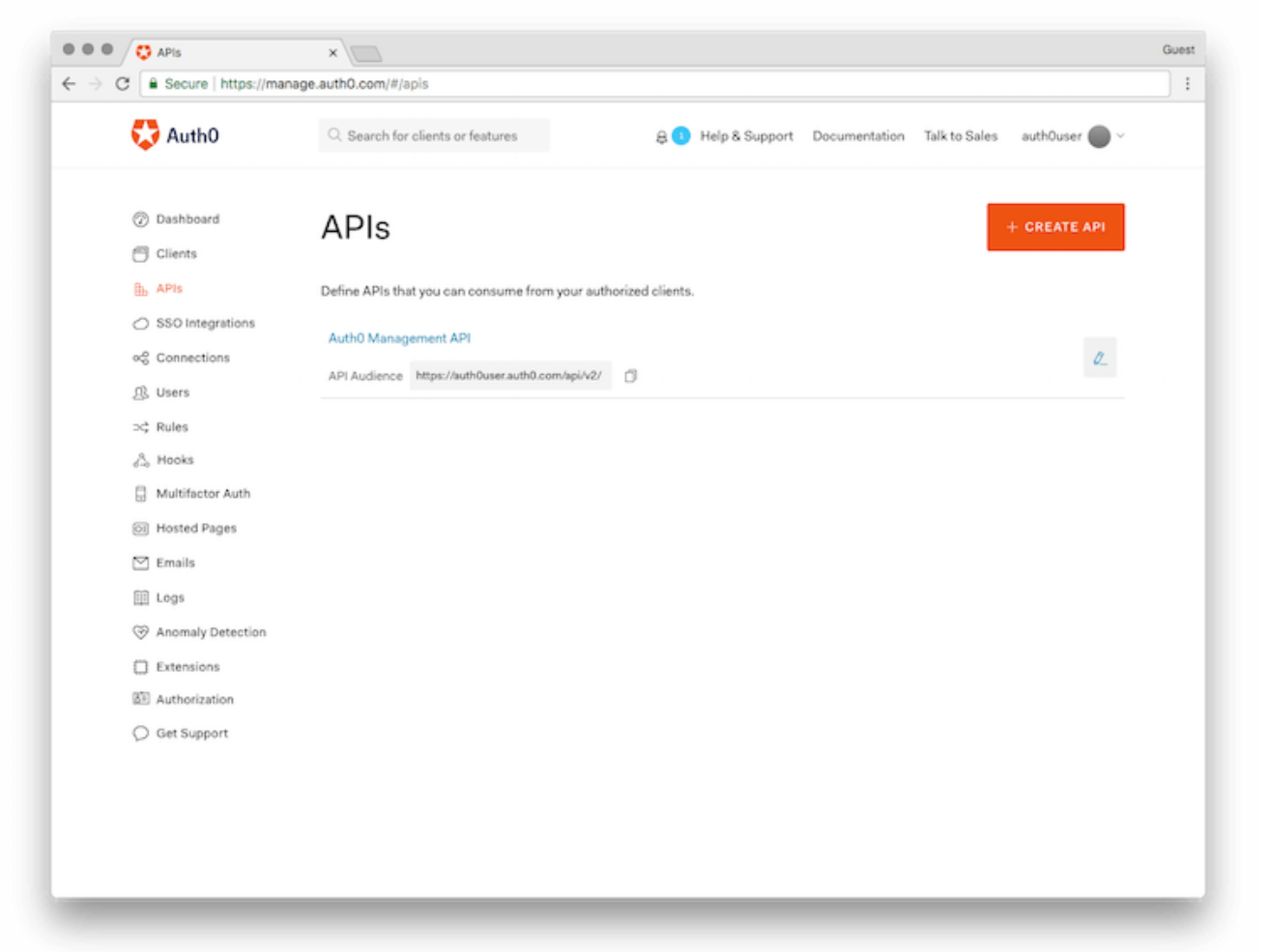
Next steps

- ✓ API Authorization
- ✓ Get Access Tokens
- ✓ JSON Web Key Sets (JWKS)

Step 1 - Configure Auth0

You will need to configure the APIs consumed by the applications that successfully authorize. You can do so using the APIs section of the Management Dashboard.

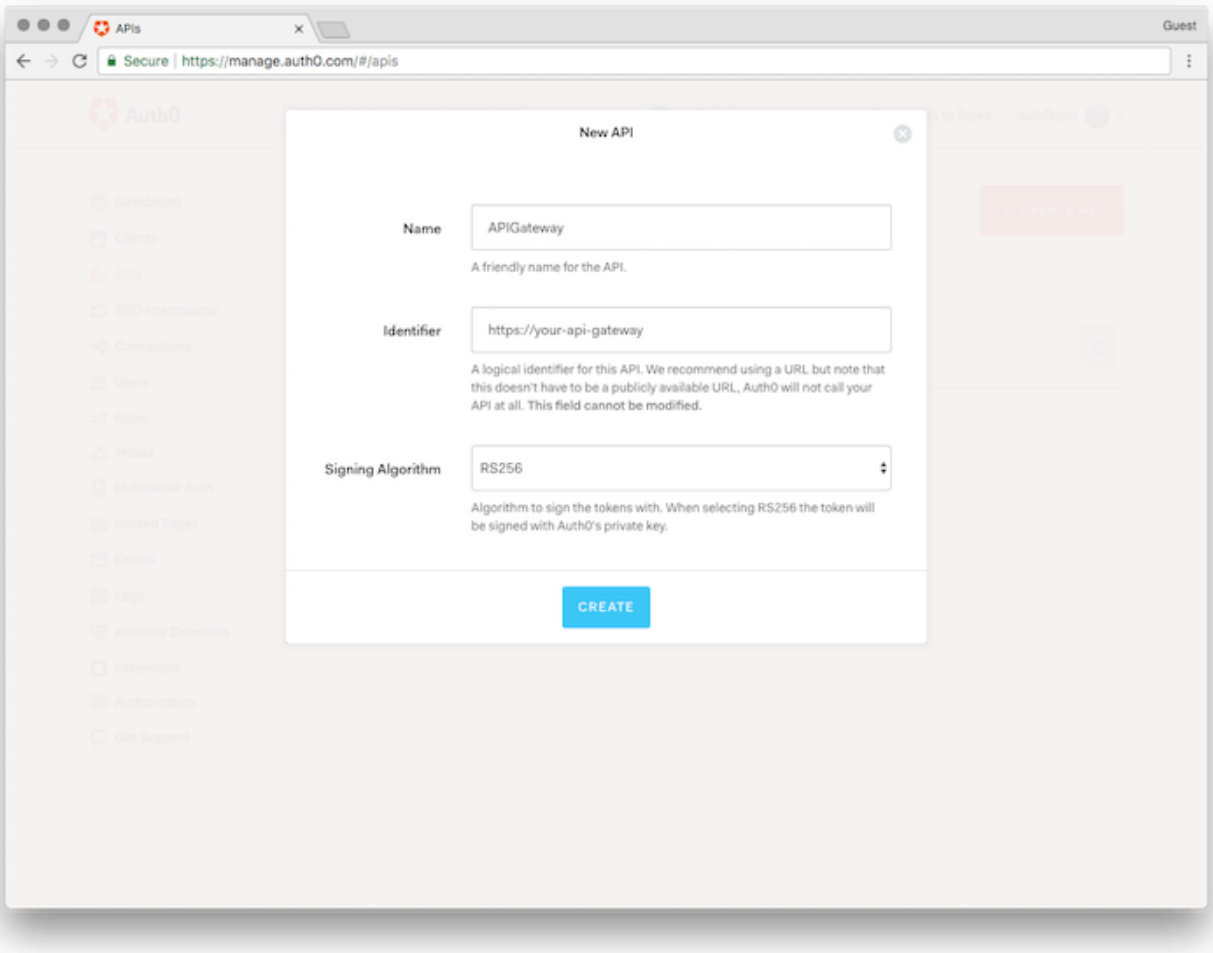
Click **Create API** to create a new API for your integration.



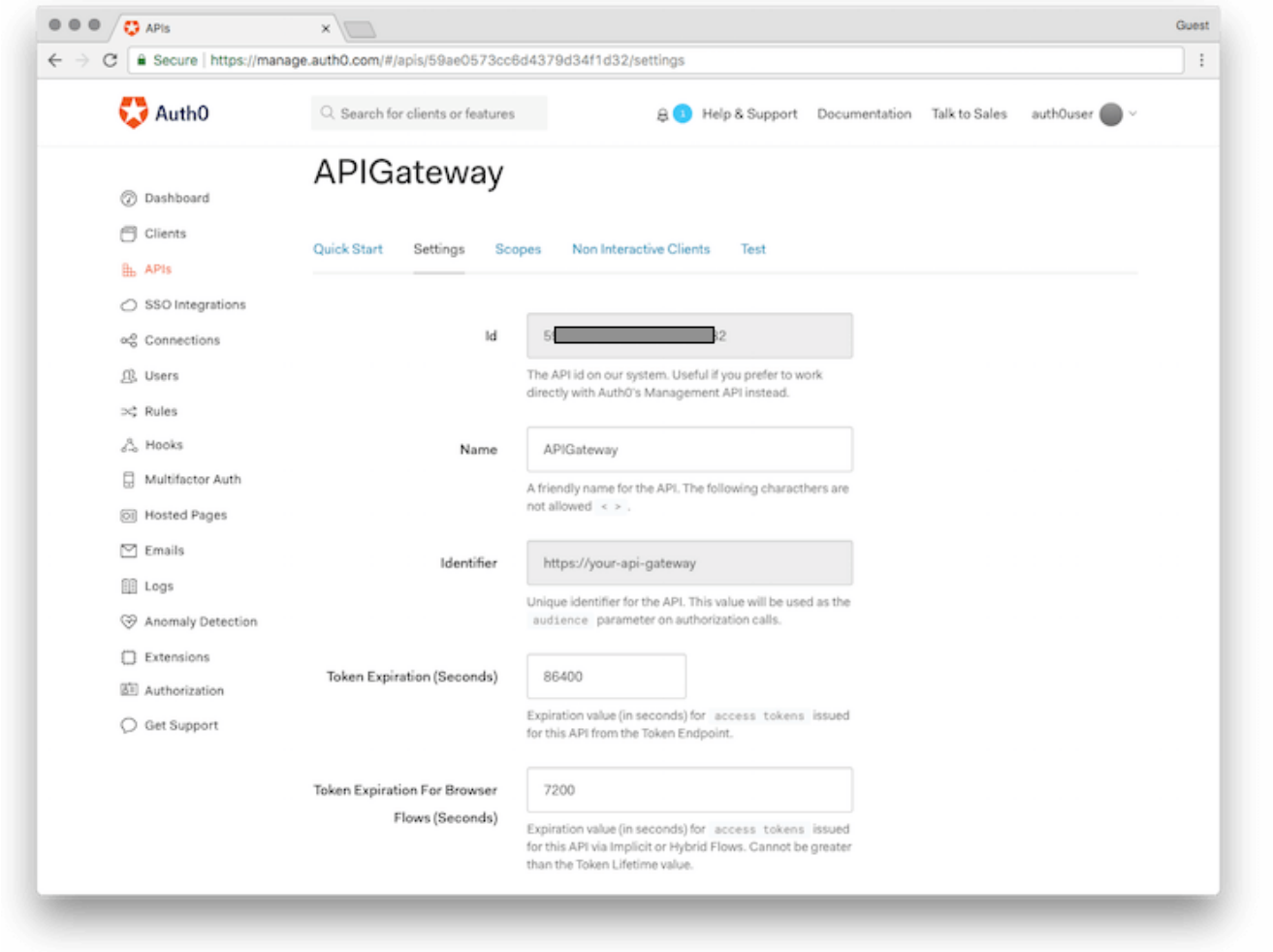
You'll be asked to provide values for the following fields:

Field	Description
Name	A friendly name for your API. This is the name you'll see in your list of Auth0 APIs.
Identifier	A logical identifier for your API (we recommend formatting this identifier like a URL https://you-api-gateway)
Signing Algorithm	The algorithm you want Auth0 to use to sign the issued Access Tokens

Click **Create** to proceed.



You can refer to the **Settings** page for the details of your newly-created API.



Creating an API also creates a Machine to Machine Application for use with the API. You can see this application listed as **Authorized** under the **Machine to Machine** Application tab. Additionally, you might want to make note of the Client ID, since you will need it in step 3 of this tutorial.

Step 2 - Import and Deploy the Gateway API

In this part of the AWS API Gateway tutorial, we will show you how to import and manage an API using API Gateway. More specifically, we will:

- ✓ Import an API into API Gateway
- ✓ Test an API import
- ✓ Deploy an API for use with any front-end applications
- ✓ Test an API deployment

In later parts of this tutorial, we will show how to secure the endpoints of this API using custom authorizers that accept Auth0 Access Tokens, as well as how we can integrate this API with our front-end JavaScript application.

Import and Configure the Pets API

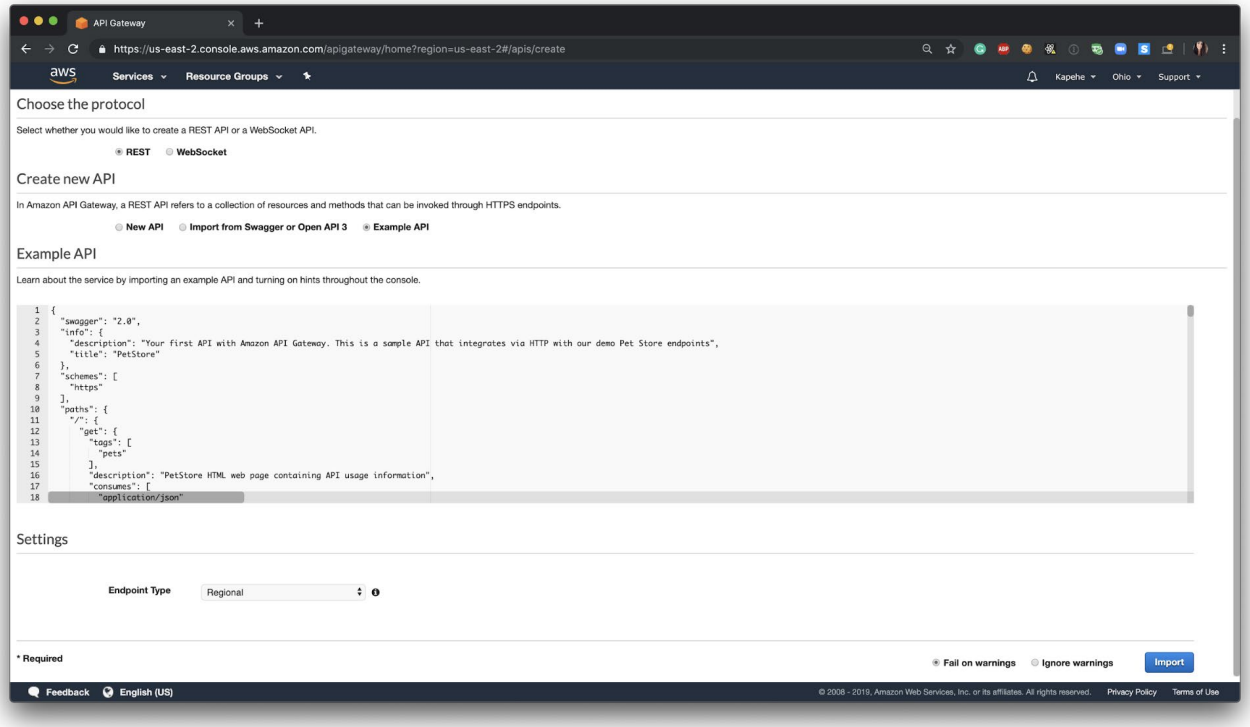
Log in to your AWS account, and using the Services drop-down located in the top navigation bar, go to the API GatewayConsole.

> ✱ ***NOTE**: If you've previously created an API, simply navigate to the API Gateway Console and click Create API. You'll be given the option to create the Example API on the Create new API form.*

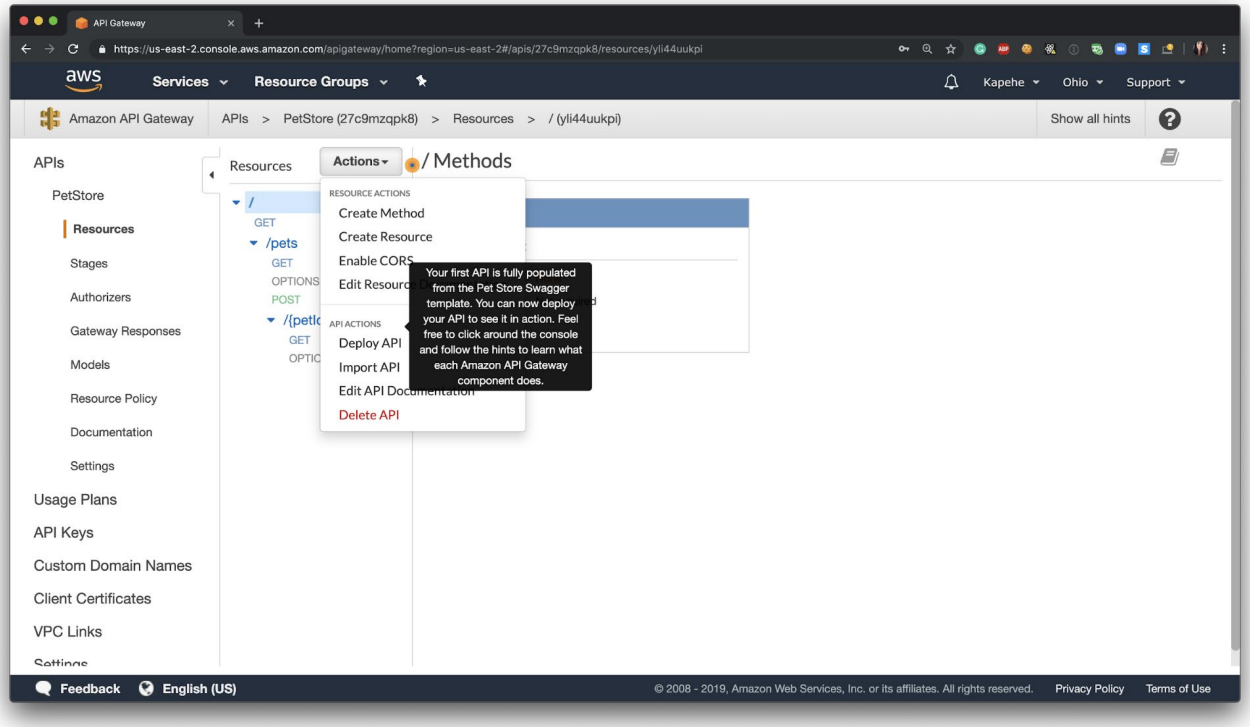
If you've never created an API using API Gateway, you'll see a screen where you can click **Get Started** to begin.

You'll see a pop-up message welcoming you to API Gateway. Click **OK** to proceed.

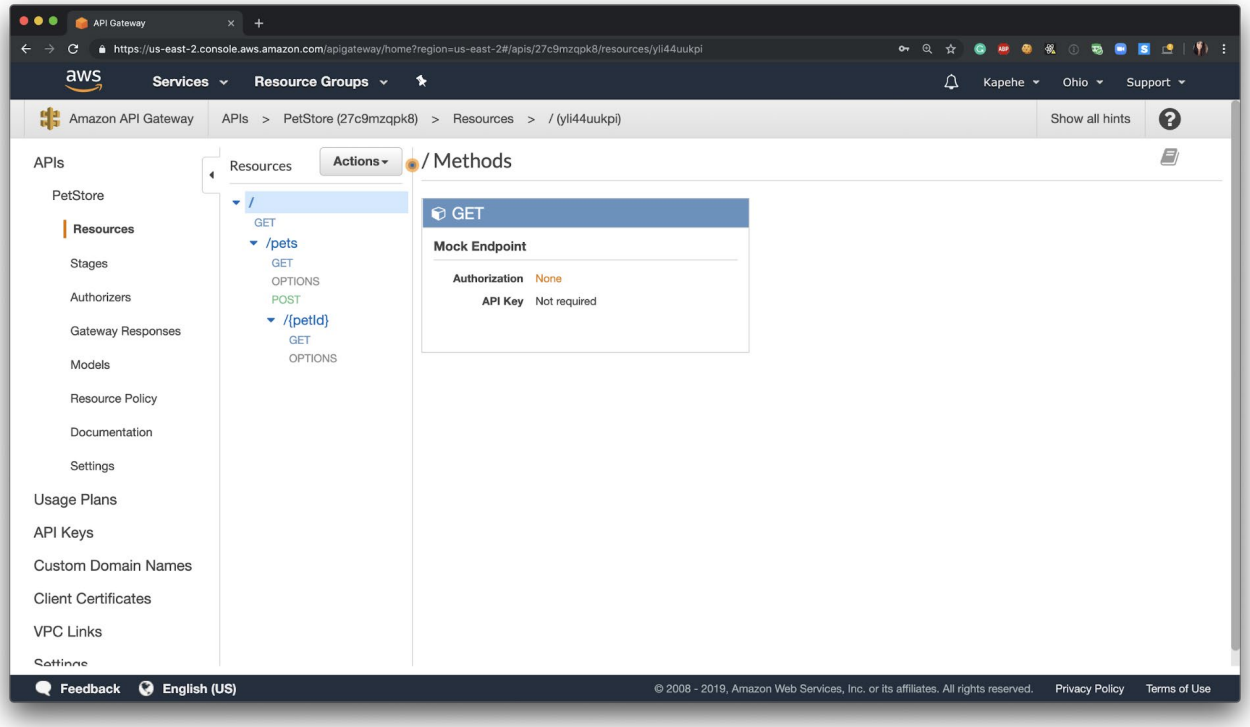
On the **Create new API** form, you'll see that **Example API** is selected by default, and there's an example API defined in the editor. We'll use this API for the rest of our tutorial, so begin the API creation process by clicking **Import**.



When done, AWS will display a message indicating that your API created and populated with the provided data.



Notice that the API already has methods associated with it (namely, GET and POST). You can view the details of a method, modify its setup, or test the method invocation by clicking the method name from the resource tree.



Test Your API

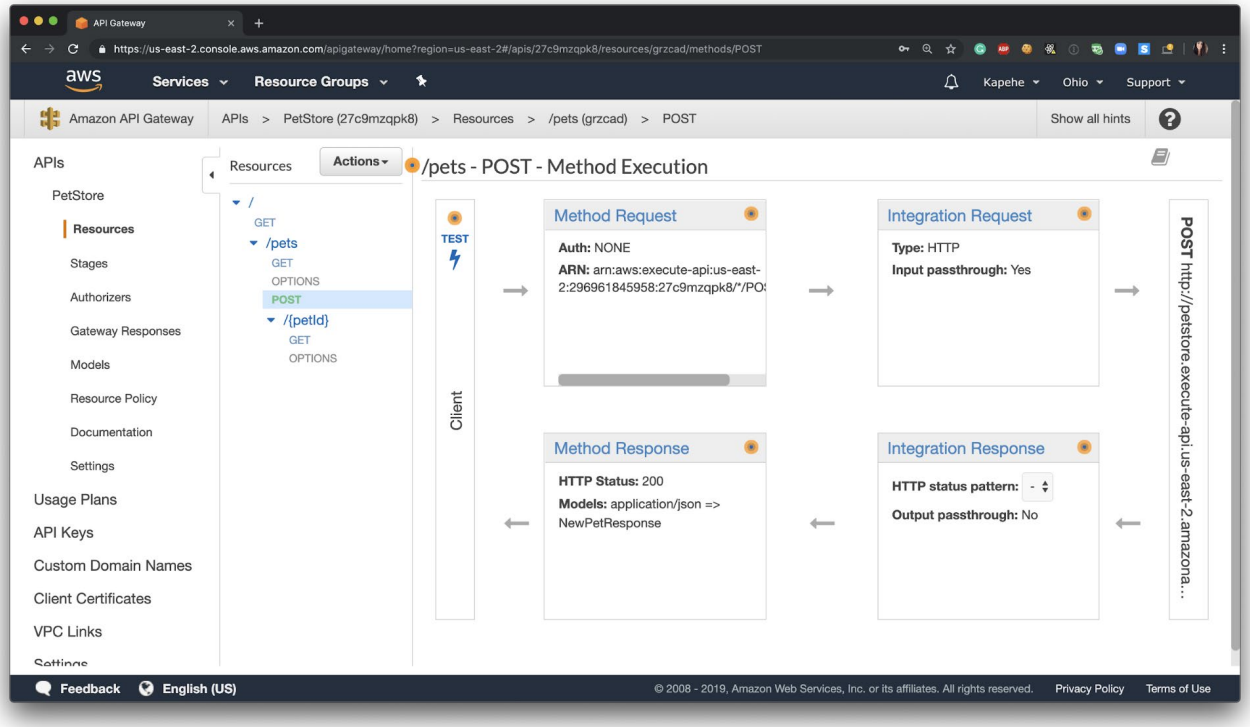
Now that you've successfully imported your API, let's run some tests to ensure that everything works as expected. This exercise will also demonstrate some of the features of the API itself.

As an example, click **POST** under `/pets`. This brings up the Method Execution window that provides an overview of the **POST** method's structure and behaviors:

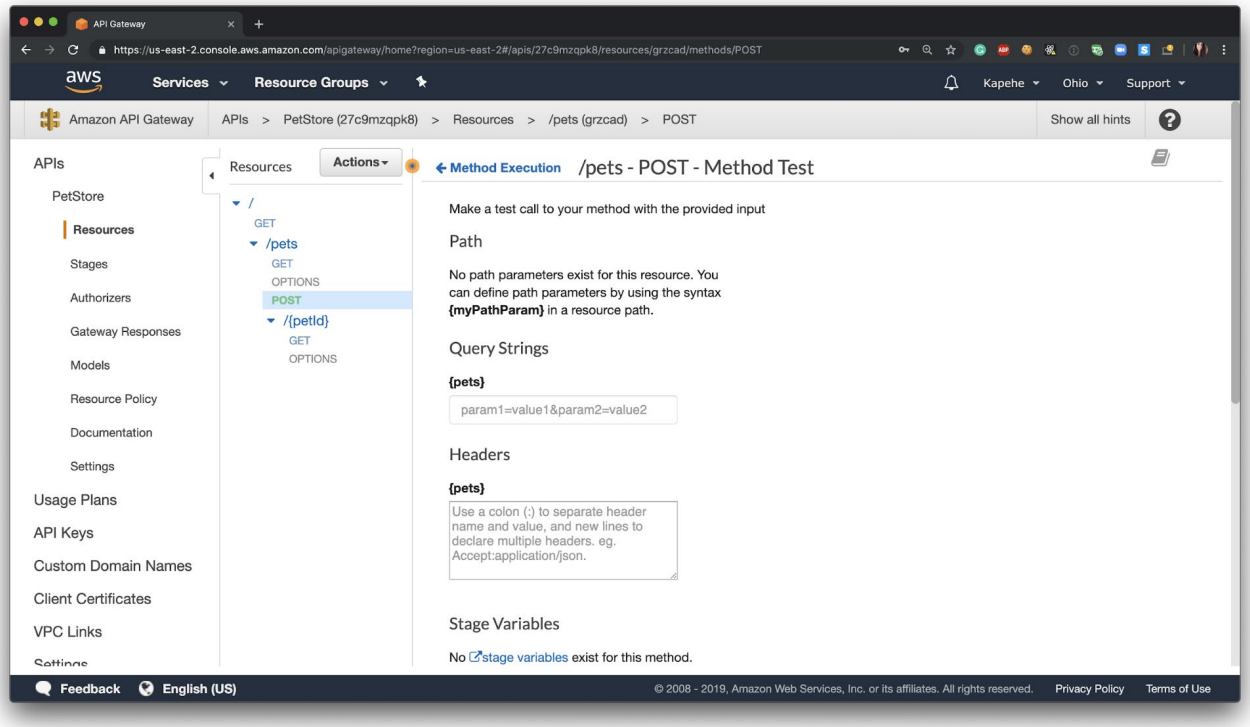
- ✓ Method Request and Method Response: the API's interface with the front-end
- ✓ Integration Request and Integration Response: the API's interface with the back-end

We can use this area to test the API we've created.

Click **Test** (shown on the **Application** sliver located in the middle of the page).



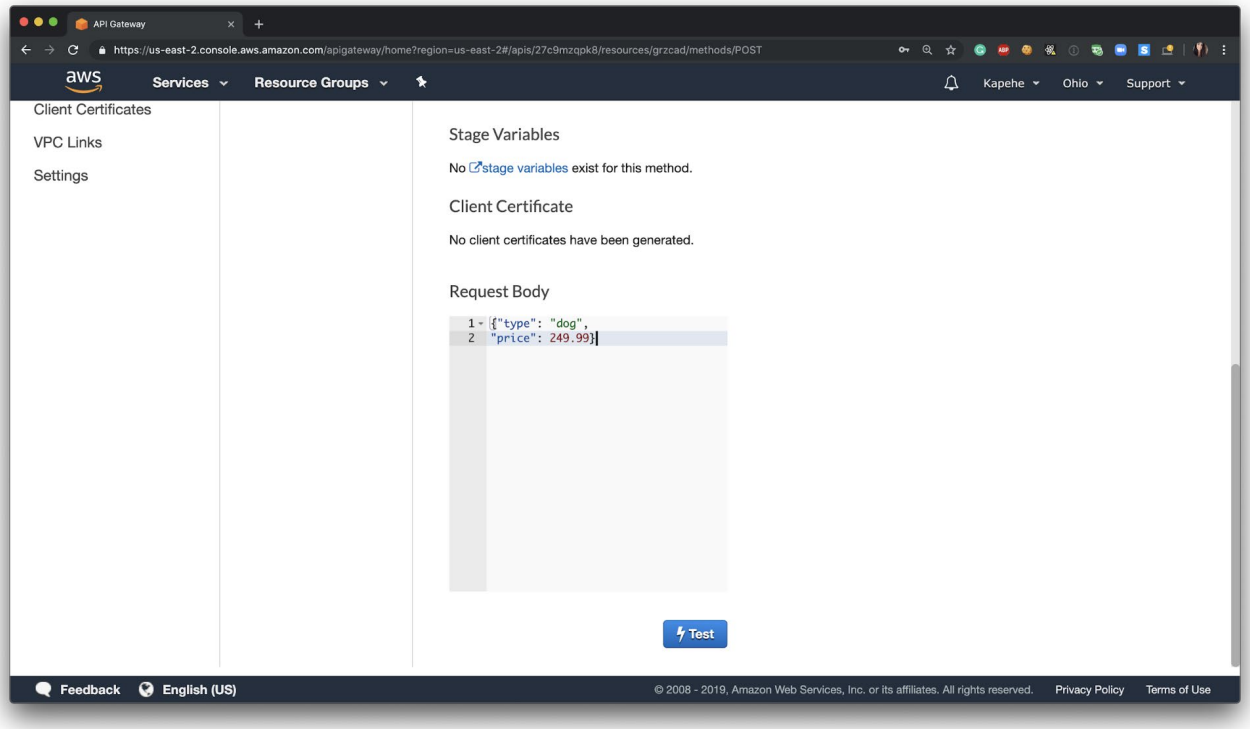
You'll be redirected to the `/pets - POST - Method Test` page.



Scroll to the bottom of the page, and provide the following snippet as the **Request Body**:

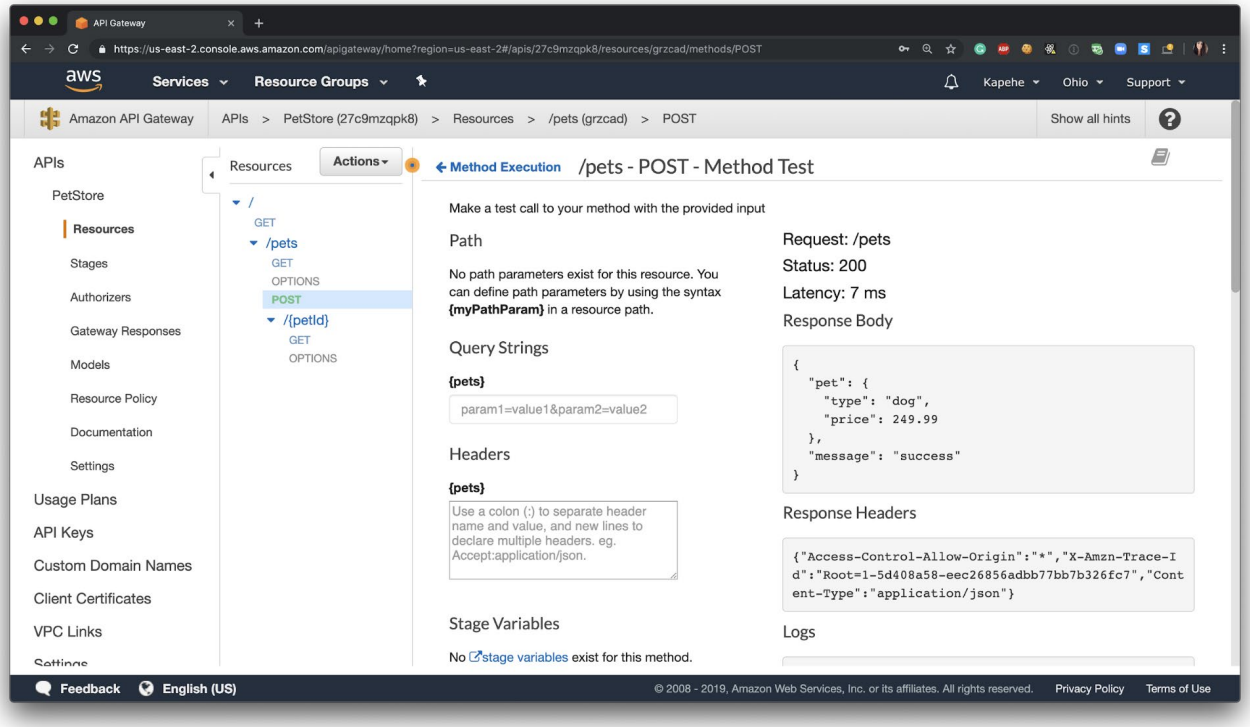
```
{"type": "dog", "price": 249.99}
```

The request body indicates the attributes of the pet we want to add to the database, as well as the cost for the pet.



Click **Test** to proceed.

You'll see the results of the test on the right side of the page.



Deploy the API

The test we just completed was done using the API Gateway console. To use the API with a different application, you'll need to deploy the API to a stage. You can do this via the Actions menu, which offers the **Deploy API** option.

You'll be asked to provide the following values:

Parameter	Value
Deployment stage	Choose [New Stage]
Stage name	Provide a name for your stage
Stage description	Provide a description for your stage
Deployment description	Provide a description for your API deployment

Once you've provided the appropriate values, click **Deploy** to proceed.

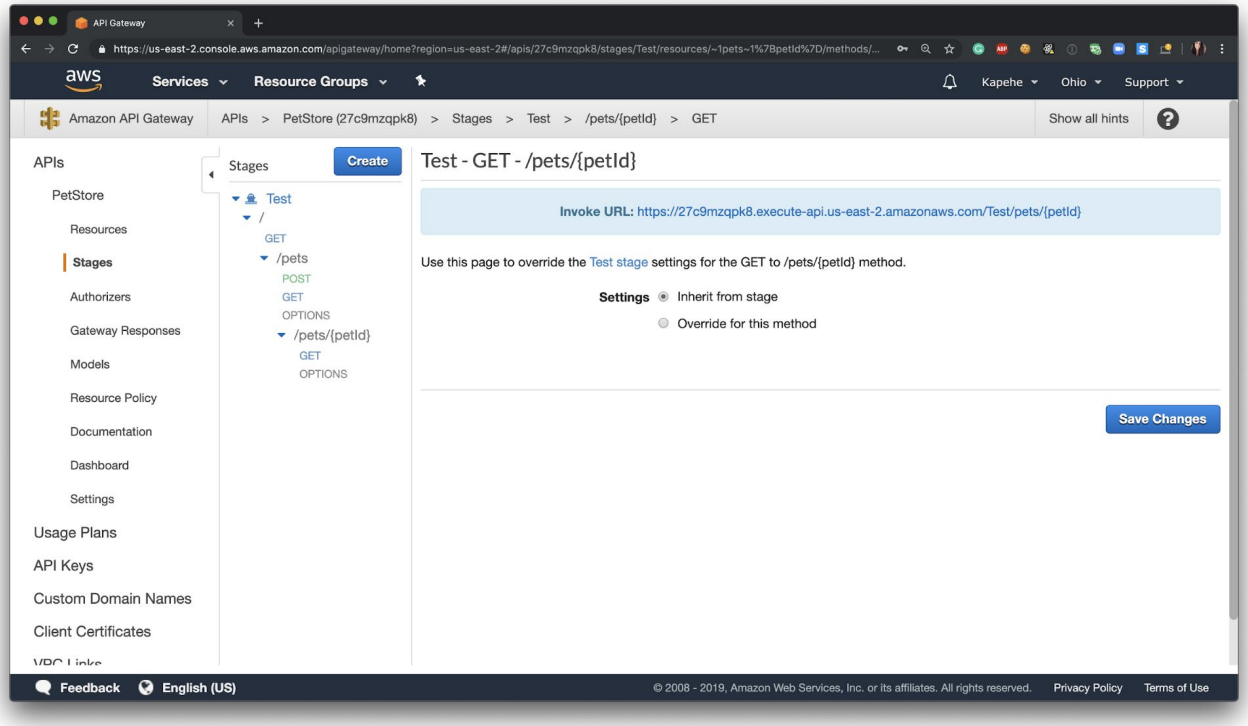
Test the Deployment

When the API has successfully deployed, you'll be redirected to the **Test Stage Editor**. You can, at this point, test the API to see if it deployed correctly.

At the top of the **Test Stage Editor** window is a blue banner with your **Invoke URL**. This is the URL used to invoke the GET endpoint of your API.

Click on the link to submit the `GET / method` request in a browser. This should result in a successful response of a webpage titled: “Welcome to your Pet Store API”.

Next, we'll make a call to **GET** under `/pets/{petId}.` In the **Stages** page, expand the tree under Test. Click **GET** under `/pets/{petId}.`



You'll see an Invoke URL displayed in the blue banner at the top of the window. The final portion, `{petID}`, stands for a path variable. Replace this variable with `1`, and navigate to the new URL using your browser. You should receive an HTTP 200 request with the following JSON payload:

```
{
    “id”: 1,
    “type”: “dog”,
    “price”: 249.99
}
```

Step 3 - Create the Custom Authorizers

In step 1, we showed you how to configure Auth0 for use with API Gateway, and in step 2 of this tutorial, we showed you how to import, test, and deploy an API using Amazon Web Services' (AWS) API Gateway. In this tutorial, we will show you how to secure this API so that only those with the appropriate authorization may access the back-end behind the API.

To do this, we will be using API Gateway's custom [request] authorizers, which allow you to authorize your APIs using bearer token authorization strategies, such as OAuth 2.0 or SAML. For each incoming request, the following happens:

1. API Gateway checks for a properly-configured custom authorizer.
2. API Gateway calls the custom authorizer (which is a Lambda function) with the authorization token.
3. If the authorization token is valid, the custom authorizer returns the appropriate AWS Identity and Access Management (IAM) policies.
4. API Gateway uses the policies returned in step 3 to authorize the request.

Prepare the Custom Authorizer

You can download a sample custom authorizer that supports Auth0-issued tokens. Afterward, you'll need to customize the files so that the custom authorizer works for your environment.

1. Unzip the folder containing the sample files you downloaded above to the location of your

choice, and navigate to the folder using the command line.

2. Within the sample folder, run `npm install` to install the Node.js packages required for deployment; AWS requires that these files be included in the bundle you will upload to AWS during a later step.

3. Configure your local environment with a `.env` file. You can copy the `.env.sample` file (while simultaneously renaming it `.env`) using cp `.env.sample .env`. Make the following changes:

Parameter	Value
JWKS_URI	The URL of the JWKS endpoint. If Auth0 is the token issuer, use https://YOUR_DOMAIN/.well-known/jwks.json
AUDIENCE	The audience value of the API you create in step 1.
TOKEN_ISSUER	The issuer of the token. If Auth0 is the token issuer, use https://YOUR_DOMAIN/. Be sure to include the trailing slash.

As an example, the text of your .env file should look something like this when complete:

```
JWKS_URI=https://YOUR_DOMAIN/.well-known/jwks.json
AUDIENCE=https://your-api-gateway
TOKEN_ISSUER=https://YOUR_DOMAIN/
```

4. Test the custom authorizer locally.

   a. First, obtain a valid JWT Access Token. There are multiple methods by which you can get one, and the method you choose depends on your application's type, trust level, or overall end-user experience.

You can get a test token for your API by going to **APIs > Your API > Test** in the dashboard. For specific details refer to Get Access Tokens.

b. Create a local `event.json` file containing the token. You can copy the sample file (`run cp event.json.sample event.json`). Replace `ACCESS_TOKEN` with your JWT token, and `methodArn` with the appropriate ARN value for the `GET` method of your API.

To get the methodArn:

Using the API Gateway Console, open the **PetStore API**.

Click **Resources** in the left-hand navigation panel.

In the middle **Resources** panel, expand the resource tree. Underneath `/pets`, click **GET**.

In the **Method Request** box, you'll see the **ARN**.

c. Run the test using npm test. The test uses the lambda-local package to test the custom authorizer using your token. If the test was successful, you'll see in the output this in the "Effect" line:

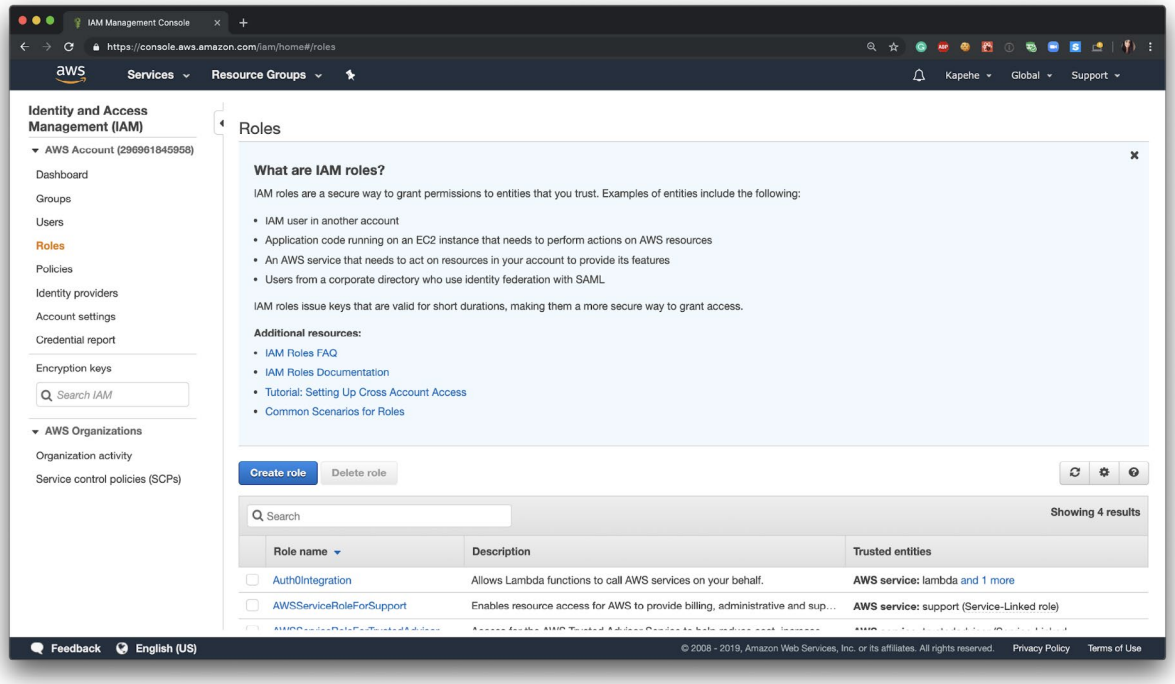
```
"Effect": "Allow",
```

Create the IAM Role

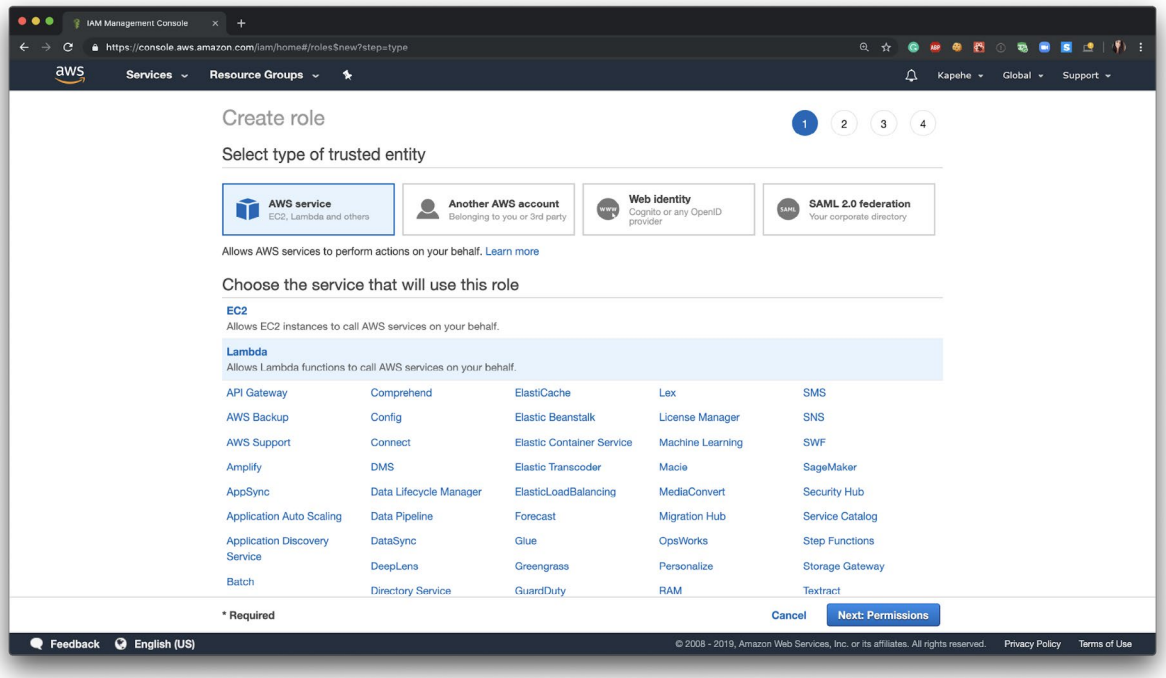
The IAM role has the required permissions to call Lambda functions; before we can proceed with our custom authorizer, we'll need to create an IAM role that can call our custom authorizer whenever API Gateway receives a request for access.

1. Log in to AWS and navigate to the IAM Console. Click Roles in the left-hand navigation bar.

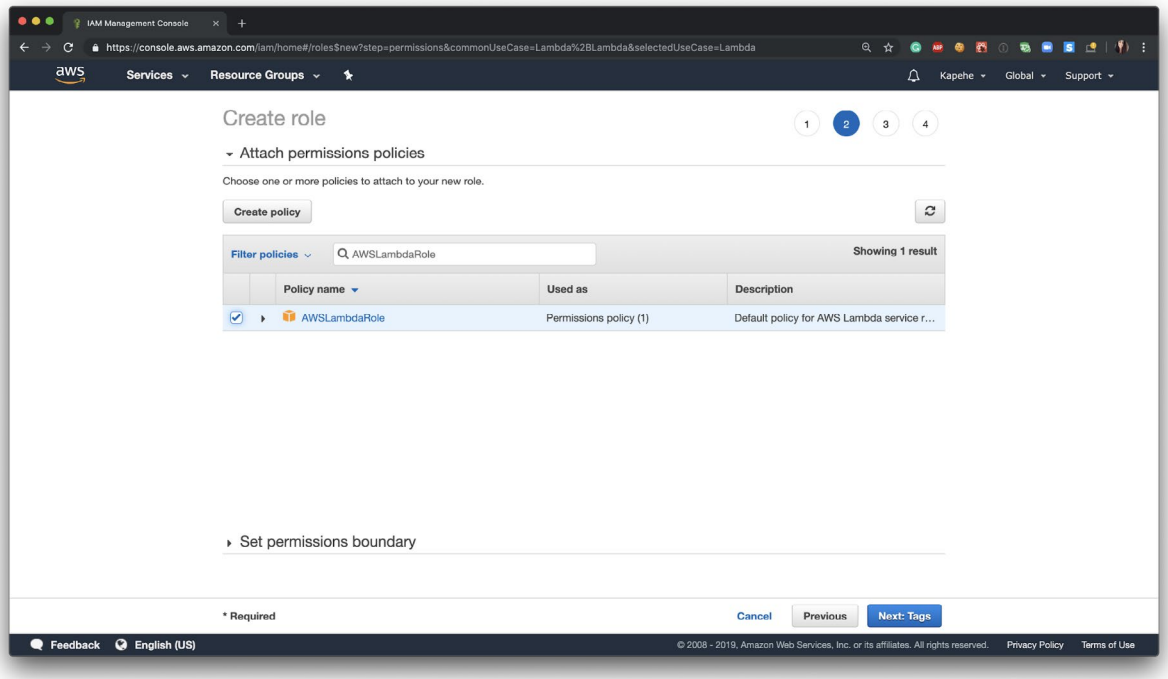
2. Click **Create role.**



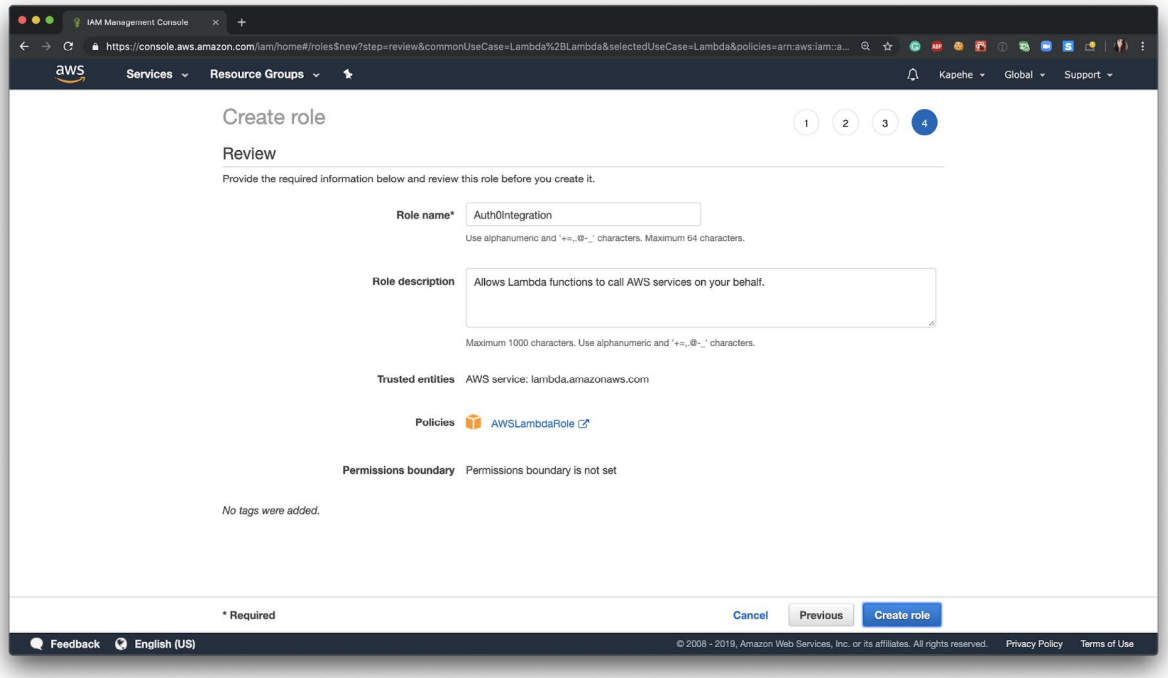
3. Keep **AWS Service Role** selected, find the **Lambda** row and click that row, then click **Next: Permissions**.



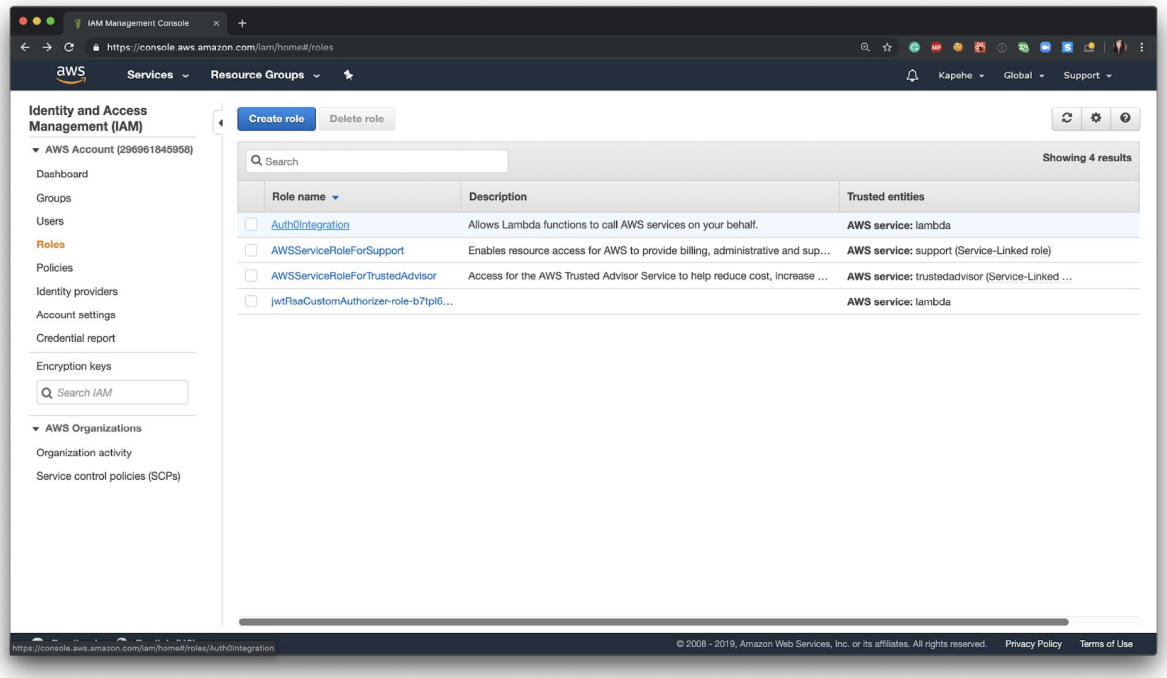
4. On the **Attach permissions policy** screen, search for and select the **AWSLambdaRole**. You can use the provided filter to narrow down the list of options. Click **Next: Tags** to proceed



5. Click **Next: Review** to skip the **Add tags (optional)** section. On **Review**, provide a Role name, such as `AuthOIntegration`. Leave the rest of the fields as is. Click **Create role**.



6. Once AWS has created your role, you'll be directed back to the **Roles** page of IAM. Select your new role.



7. On the **Summary** page for the role you've just created, click on to the Trust **relationships tab**.

8. Click **Edit trust relationship**, and populate the **Policy Document** field with the following JSON snippet:

```
{
   "Version": "2012-10-17",
   "Statement": [
       {
       "Effect": "Allow",
       "Principal": {
           "Service": [
           "apigateway.amazonaws.com",
           "lambda.amazonaws.com"
           ]
       },
       "Action": "sts:AssumeRole"
       }
   ]
}
```

Click **Update Trust Policy**.

9. You'll be redirected back to the **Summary** page. Copy down the **Role ARN** value for later use.

Create the Lambda Function and Deploy the Custom Authorizer

1. First, you'll need to create a bundle that you can upload to AWS by running `npm run bundle` in your **jwt-rsa-aws-custom-authorizer-master download** from earlier. This generates a `custom-authorizer.zip` bundle containing the source, configuration, and node modules required by AWS Lambda.

2. Navigate to the Lambda console, and click **Create function**, and ensure **Author from scratch** is highlighted.

3. Scroll down on the Create function page and populate the Function name field and leave Runtime as is:

Parameter	Value
Name	A name for your Lambda function, such as jwtRsaCustomAuthorizer
Runtime	Select Node.js 8.10

4. In the Permissions section, set the following values and click Create Function:

Parameter	Value
Role	Choose an existing role
Existing Role	Select the IAM role you created in the steps above.

5. Next, you will be redirected to the **jwtRSACustomAuthorizer page**. Under **Function** code, find **Code entry type**, and select **Upload a .zip file**. Click **Upload** and select the `custom-authorizer.zip` bundle you created earlier. Keep **Handler** as `index.handler`.

6. Then, create the following three Environment variables. Note that this information is identical to that which is in the .env file:

Parameter	Value
TOKEN_ISSUER	The issuer of the token. If Auth0 is the token issuer, use https://YOUR_DOMAIN/
JWKS_URI	The URL of the JWKS endpoint. If Auth0 is the token issuer, use https://YOUR_DOMAIN/.well-known/jwks.json
AUDIENCE	The audience value of the API you created in step 1.

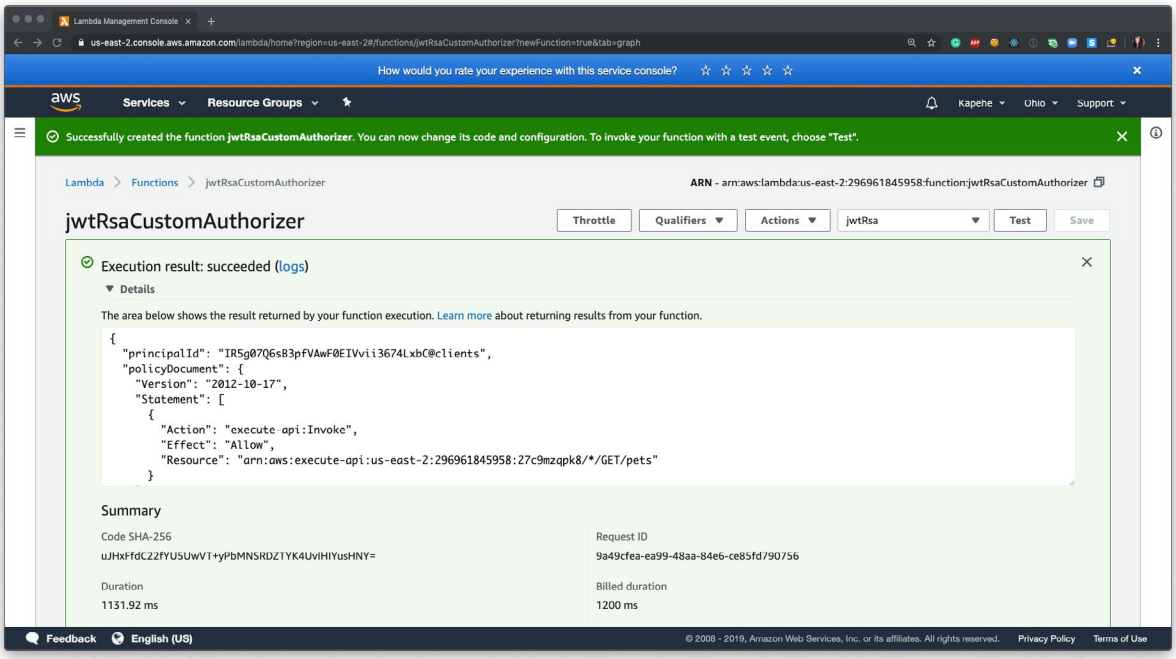
7. In the **Basic settings** area set Timeout to 30 sec, then click **Save**. AWS will inform you if you have successfully created your function.

8. Test the **Lambda** function you just created. Click **Test** in the top right corner.

9. Copy the contents of your `event.json` file into the **Configure test event** JSON (you can use the default "Hello World" template). Give it an **Event name**, *jwtRsa*.

```
{
 "type": "TOKEN",
 "authorizationToken": "Bearer <TOKEN HERE>",
 "methodArn": "arn:aws:execute-api:us-east-2:296961845958:27c9mzqpk8/*/
GET/pets"
}
```

Click **Create** and then click **Test** again. If the test was successful, you'll see the “Execution result” be successful:



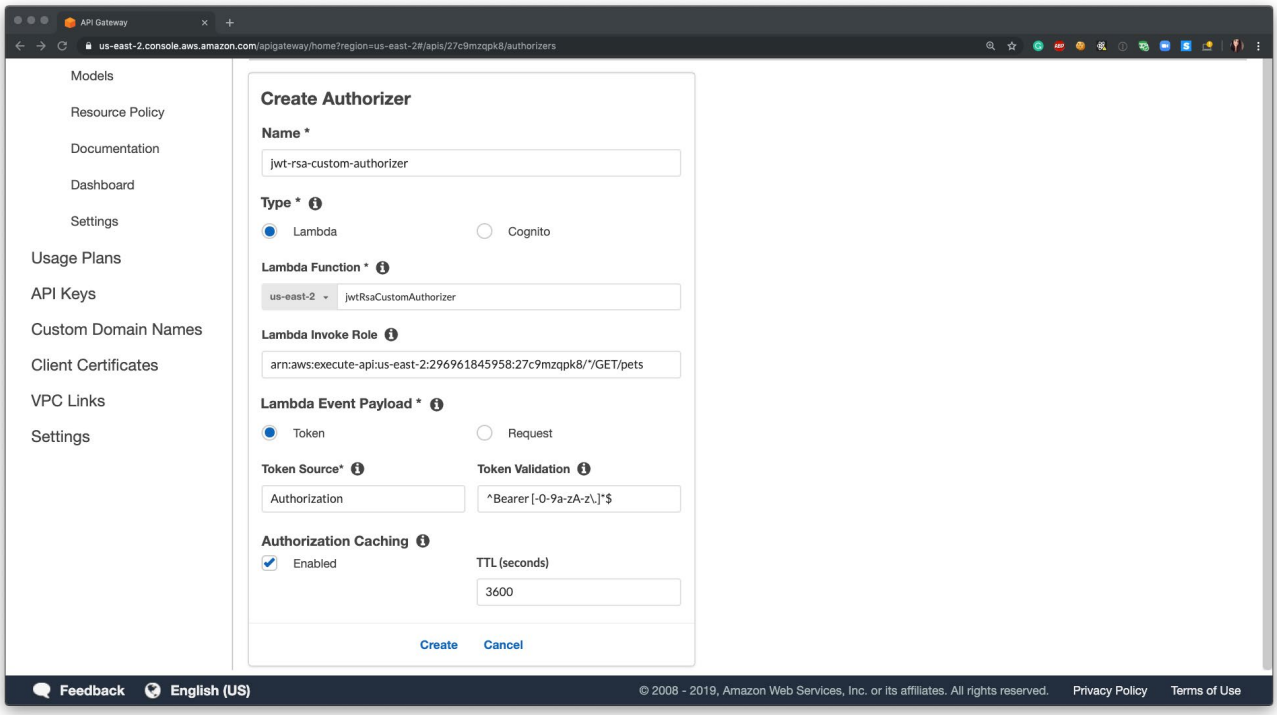
Configure API Gateway Custom Authorizer

Return to the API Gateway Console and open the **PetStore** API we created earlier.

Using the left-hand navigation bar, open **Authorizers**. If this is the first authorizer you've created, you'll see the **Create New Authorizer** configuration screen by default. If not, you can bring up this screen by clicking Create > Custom Authorizer on the center panel.

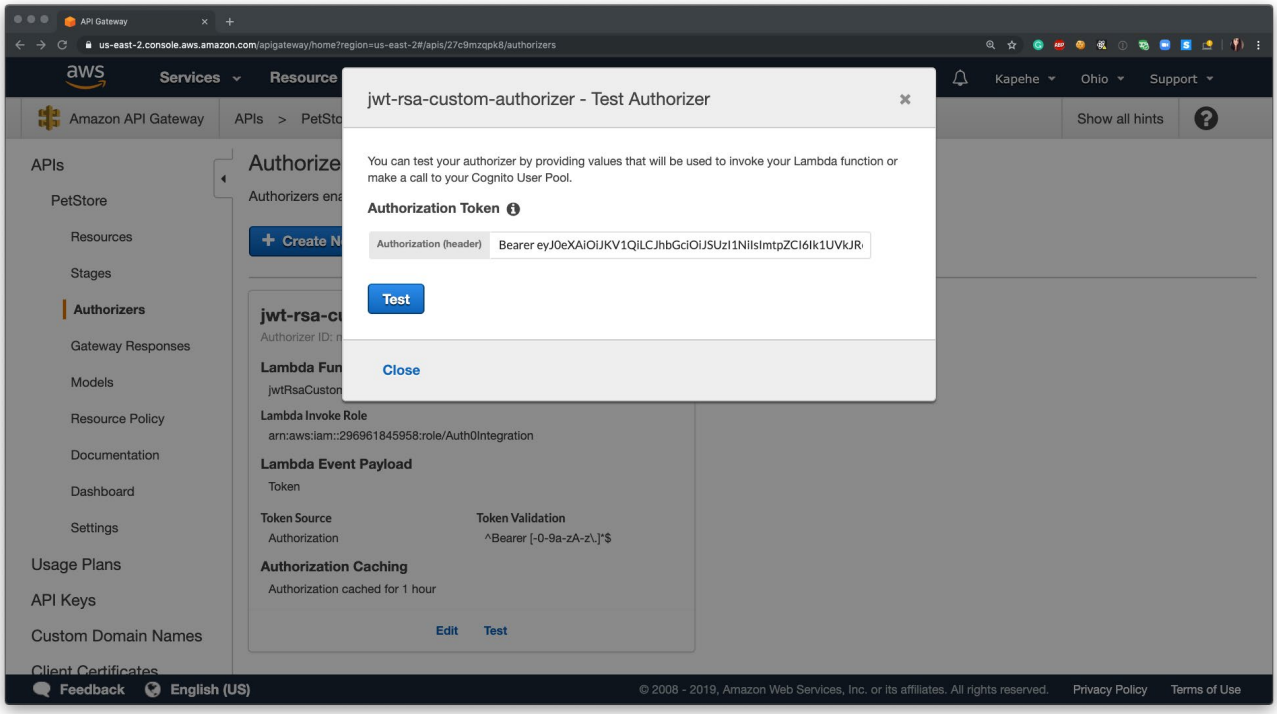
Set the following parameters and click Create:

Parameter	Value
Name	jwt-rsa-custom-authorizer
Type	Lambda
Lambda Function	us-east-2 and jwtRsaCustomAuthorizer
Lambda Invoke Role	The IAM Role ARN you copied above
Lambda Event Payload	Token
Token Source and Validation	Authorization and ^Bearer [-0-9a-zA-z\.]*$
TTL (seconds)	3600

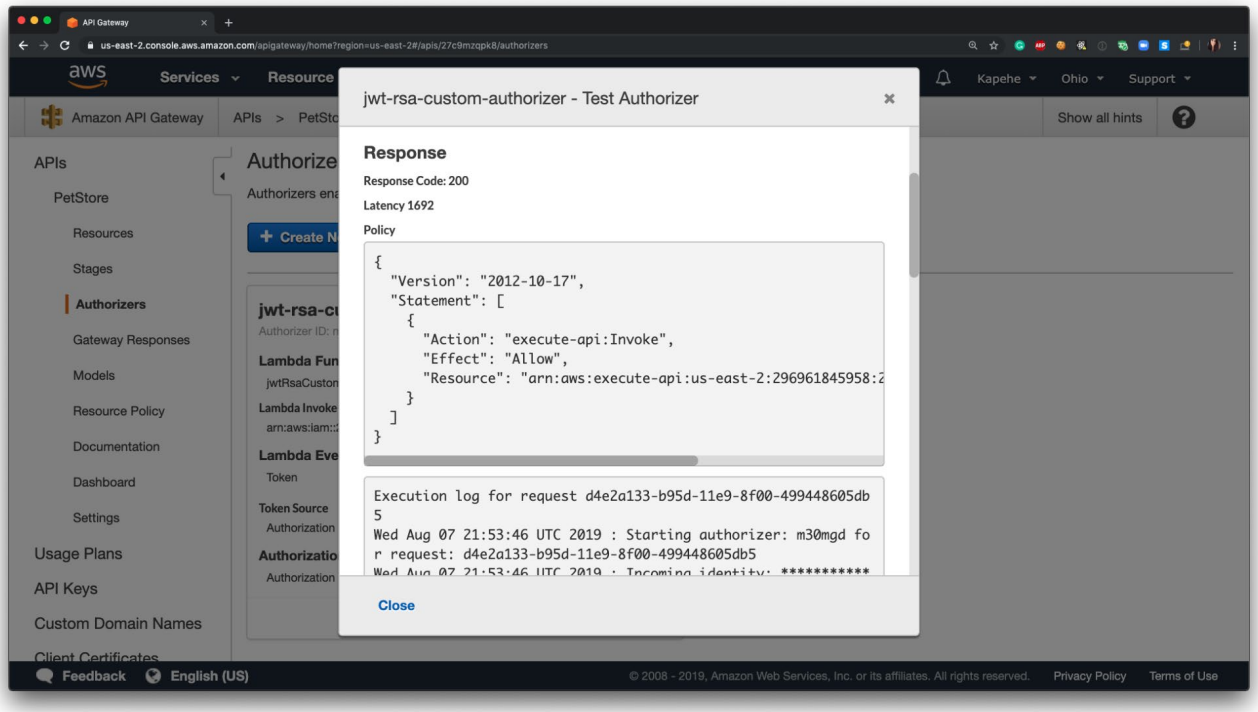


After AWS creates the authorizer and the page refreshes, click **Test** at the bottom of the screen.

You can test your authorizer by providing the Auth0 token (`Bearer ey...`) you've previously used and clicking **Test**.



If the test was successful, you'll see a response similar to the following:



Summary

In this part of the API Gateway tutorial, we configured the custom authorizer we'll use to handle access requests. To do this, we:

- ✓ Prepared a bundle containing the code that will be used by the Lambda function using the Auth0 sample
- ✓ Created the IAM role that will call the Lambda function
- ✓ Created the Lambda function using the custom bundle created in step 1

✓ Created the API Gateway custom authorizer using the Lambda function we created in step 3

In the next part of the tutorial, we will use the custom authorizer we created.

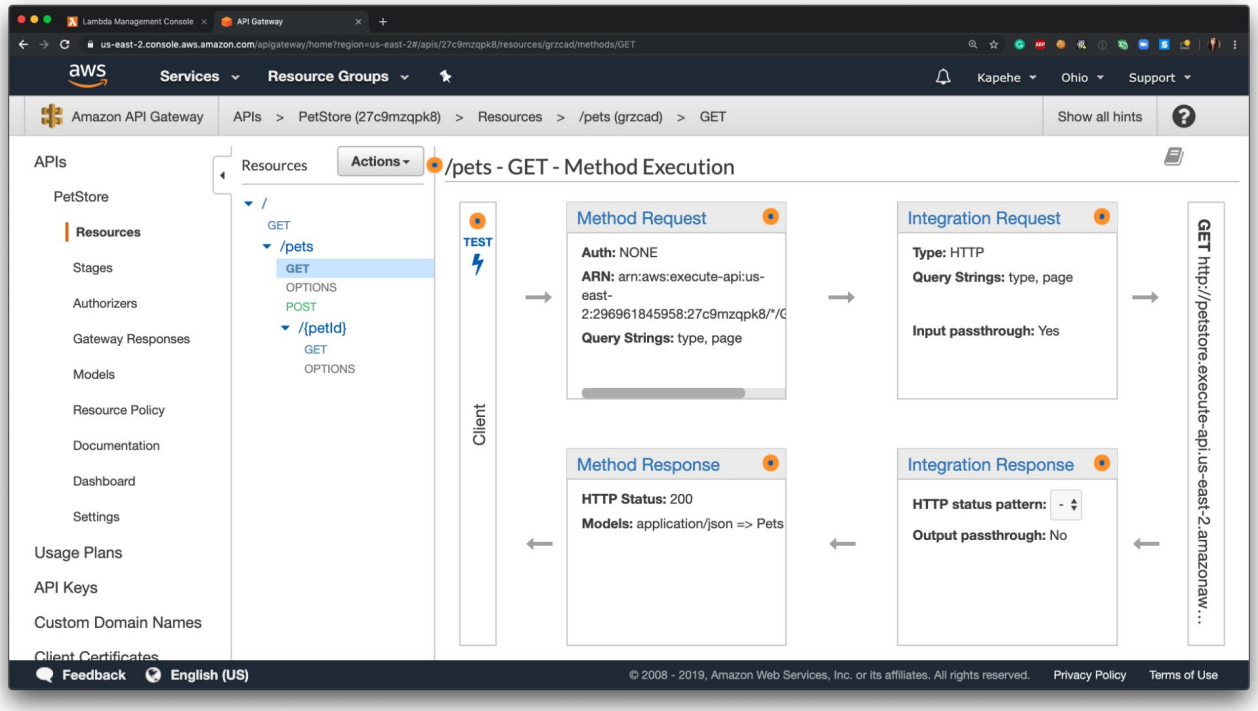
Step 4 - Secure the API Using Custom Authorizers

In step 1, you configured Auth0 for use with API Gateway, in step 2, you configured an API using API Gateway, and in step 3, you created the custom authorizer that can be used to retrieve the appropriate policies when your API receives an access request. In this part of the tutorial, we will show you how to use the custom authorizer to secure your API's endpoints.

✱ ***NOTE**: Custom authorizers are set on a method by method basis; if you want to secure multiple methods using a single authorizer, you'll need to repeat the following instructions for each method.*

Configure API Gateway Resources to Use the Custom Authorizer

Open the PetStore API we created in step 2 of this tutorial. Under the Resource tree in the center pane, select the GET method under the `/pets` resource, then click **Method Request**.

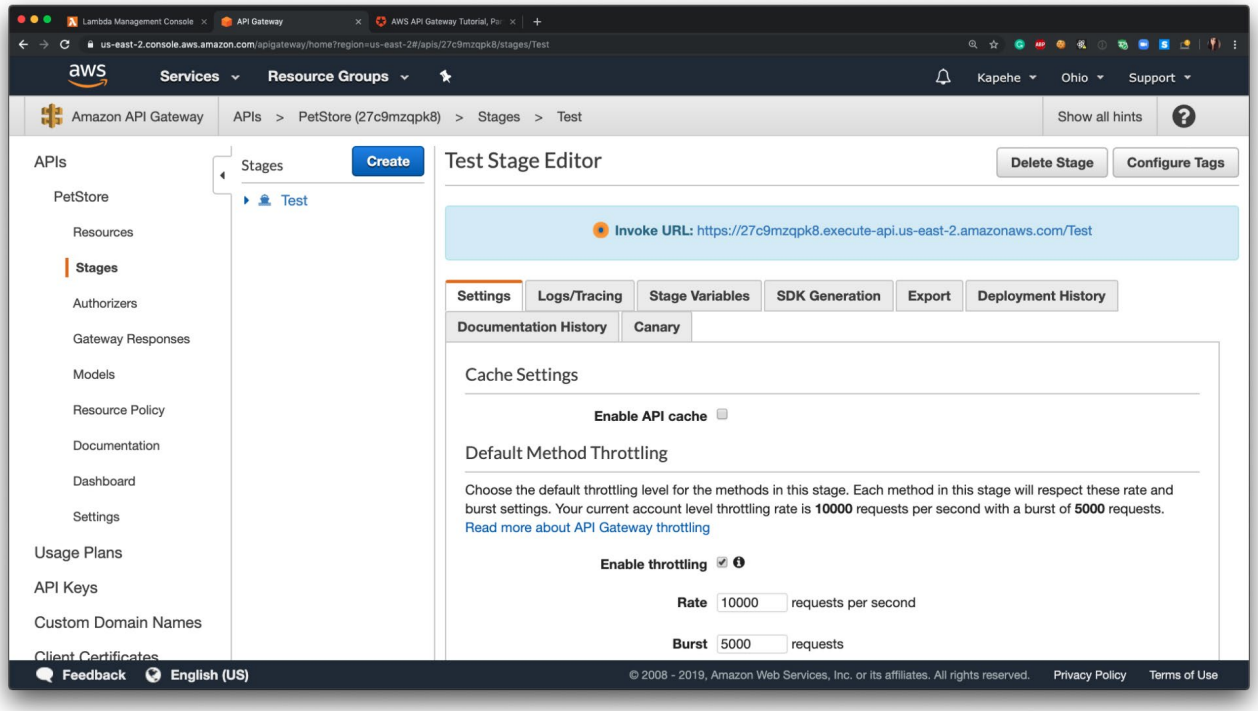


Under **Settings**, click the pencil icon to the right of **Authorization** and choose the `jwt-rsa-custom-authorizer` custom authorizer you created in step 3. Click the checkmark icon to save your choice of the custom authorizer. Make sure the API Key Required field is set to `false`.

Deploy the API

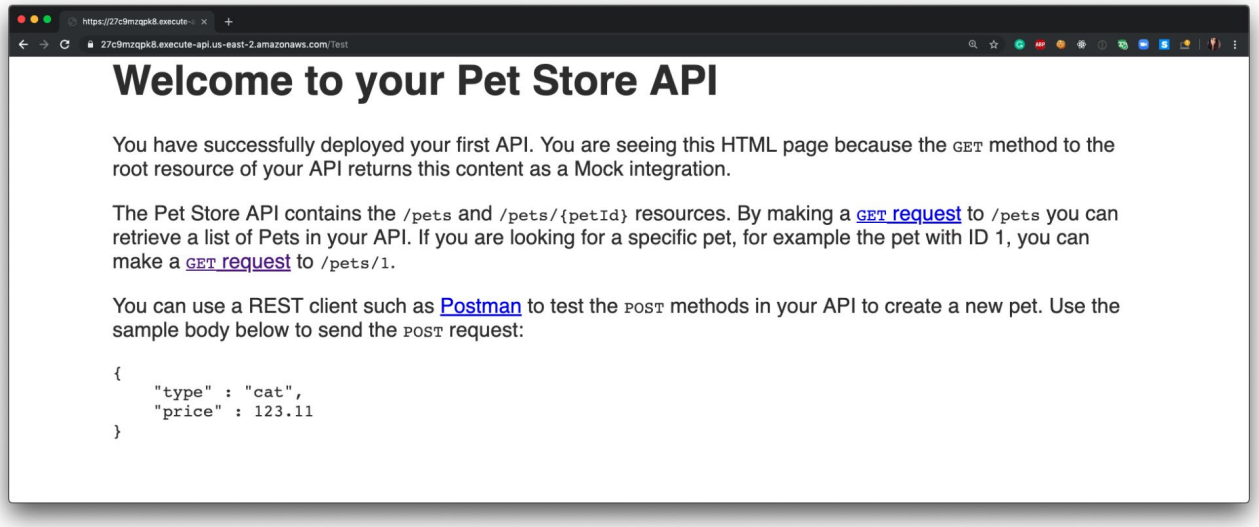
To make your changes public, you'll need to deploy your API.

If successful, you'll be redirected to the Test Stage Editor. Copy down the Invoke URL provided in the blue ribbon at the top since you'll need this to test your deployment.



Test Your Deployment

You can test your deployment by making a `GET` call to the **Invoke URL** you copied in the previous step. You should see “Welcome to your Pet Store API”:



Summary

In this tutorial, you have

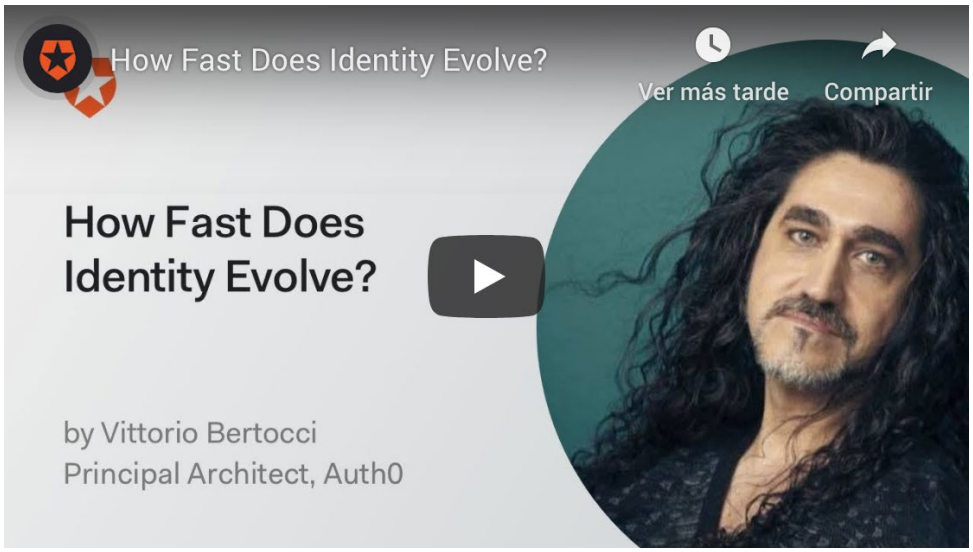
- ✓ Configured Auth0 for use with API Gateway
- ✓ Imported an API for use with API Gateway
- ✓ Created a custom authorizer to secure your API's endpoints, which required working with AWS IAM and Lambda
- ✓ Secured your API with your custom authorizer

Additional resources

Auth0 maintains a robust library of documentation that's regularly upgraded. A few immediately relevant docs include:

- ✓ Integrate Auth0 with Amazon Web Services (AWS)
- ✓ Integrate Auth0 with Amazon Cognito
- ✓ Configure Single Sign-On with the AWS Console

Best practices & project planning



Several of our most seasoned identity professionals recently put together a growing set of best practices, including project planning advice. These docs will be continually updated to keep up the rapidly evolving world of identity.

B2C Cookbook

Docs - https://auth0.com/docs/architecture-scenarios/b2c

Deck - https://docs.google.com/presentation/d/17OalYMS2P8lDiggKvMg6Ta0sN3RgS9F5m48j0nHwktM/edit#slide=id.g591ba63052_0_0

- ✓ CIAM Planning Guide
- ✓ CIAM (B2C) Introduction
  - Architecture
  - Provisioning
  - Authentication
  - Branding
  - Profile Management
  - Authorization
  - Logout
  - Deployment Automation
  - Quality Assurance
  - Operations

B2B Cookbook

Docs - https://auth0.com/docs/architecture-scenarios/b2b

Deck - https://docs.google.com/presentation/d/1aqvujzFwV39KXlzpWbLSIhScgv4K97rtqwQyhPY

wHTM/edit#slide=id.g591ba63052_0_0

- ✓ B2B IAM Planning Guide
- ✓ B2B IAM Introduction
  - Architecture
  - Provisioning
  - Authentication
  - Branding
  - Profile Management
  - Authorization
  - Logout
  - Deployment Automation
  - Quality Assurance
  - Operations
- ✓ Multi-tenancy Overview
  - Multi-tenancy - Isolated users by Organization

Questions?

For more information about Auth0 on AWS and how together we solve the complexity inherent in Identity & authentication visit Auth0.com or reach out to an Auth0 resource.