**API Gateway Deployments and Stages**

**Understanding the Pipeline:**

* Making changes to your API in API Gateway isn't enough for them to be live.
* You need to deploy those changes to a specific **stage** for them to take effect.

**Stages: Environments for your API**

* Stages act like different environments for your API.
* You can have as many stages as you need (common names: dev, test, prod).
* Each stage can have its own configuration, allowing you to tailor settings for each environment.

**Benefits of Stages:**

* **Controlled Rollouts:** Deploy changes to a staging environment (like dev or test) first to identify and fix issues before pushing them to production.
* **Rollback Capability:** API Gateway keeps a history of deployments for each stage. If something goes wrong, you can easily roll back to a previous version.

**Key Takeaways:**

* Deployments are how you push changes from your API definition to a specific stage.
* Stages provide a way to manage different environments for your API and control rollouts.

**Naming Conventions:**

You can use any naming convention for your stages, but common choices include "dev," "test," and "prod" to represent development, testing, and production environments respectively.

**Stage Variables in API Gateway**

**Think Environment Variables, But for Stages**

* Stage variables are similar to environment variables, but specific to stages in API Gateway.
* They allow you to manage frequently changing configuration values without modifying your API definition.

**Where to Use Them:**

* **Lambda Function ARN:** Reference the Lambda function you want to invoke based on the stage.
* **HTTP Endpoint:** Define the backend endpoint URL based on the stage (e.g., dev vs. prod).
* **Mapping Templates:** Use stage variables within mapping templates to dynamically adjust parameters.

**Real-World Applications:**

* **Environment-Specific Endpoints:** Configure your API to talk to different HTTP endpoints depending on the stage (dev, test, prod).
* **Lambda Configuration:** Pass stage-specific configurations to your Lambda functions through mapping templates.
* **Access in Lambda:** Stage variables are accessible within the context object of your Lambda functions using the format ${stageVariables.variableName}.

**Remember the Format:**

* To reference a stage variable within your API configuration, use the format ${stageVariables.variableName}.

In essence, stage variables provide a flexible way to manage configuration based on the deployment stage of your API Gateway.

**API Gateway Stage Variables and Lambda Aliases**

|  |  |
| --- | --- |
| Feature | Description |
| **Stages** | Represent environments (Prod, Dev, Test) for your API. |
| **Stage Variables** | Configuration values specific to a stage (like environment variables). |
| **Lambda Aliases** | Pointers to specific versions of your Lambda function. |

**Scenario:**

Consider a scenario with three stages (Prod, Dev, Test) and corresponding Lambda aliases (PROD, DEV, TEST). Traffic weights are also assigned (Prod: 95%, Dev: 5%, Test: 100%).

**How it Works:**

1. **Stage Variable Creation:** Create stage variables for each stage that point to their respective Lambda aliases (e.g., ProdStageVariable = PROD).
2. **API Gateway Configuration:** Use the stage variable in your API configuration to reference the Lambda function (e.g., ${stageVariables.lambdaAlias}).
3. **Automatic Routing:** API Gateway automatically routes requests based on the stage and its corresponding alias:
   * Prod (95%): Routes to PROD alias.
   * Dev (5%): Routes to DEV alias.
   * Test (100%): Routes to TEST alias.

**Benefits:**

* **Dynamic Configuration:** Manage different configurations (Lambda versions) per stage using stage variables.
* **Simplified Deployments:** Update Lambda aliases for new versions without modifying the API definition.
* **Reduced Downtime:** Seamlessly switch between Lambda versions during deployments.

**Key Point:**

* Stage variables are accessible in your Lambda function using ${stageVariables.variableName}.

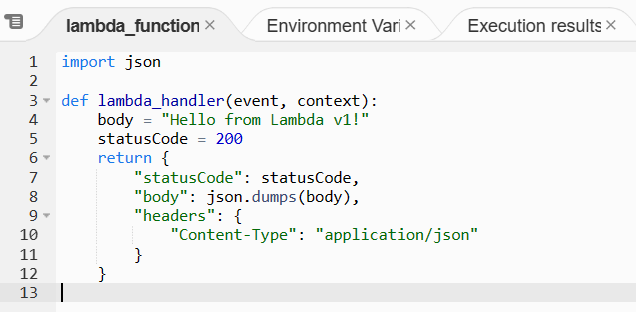
**Overall Advantage:** This approach allows you to manage Lambda function versions and configurations based on stages without modifying your core API Gateway definition, leading to more streamlined deployments and reduced downtime.

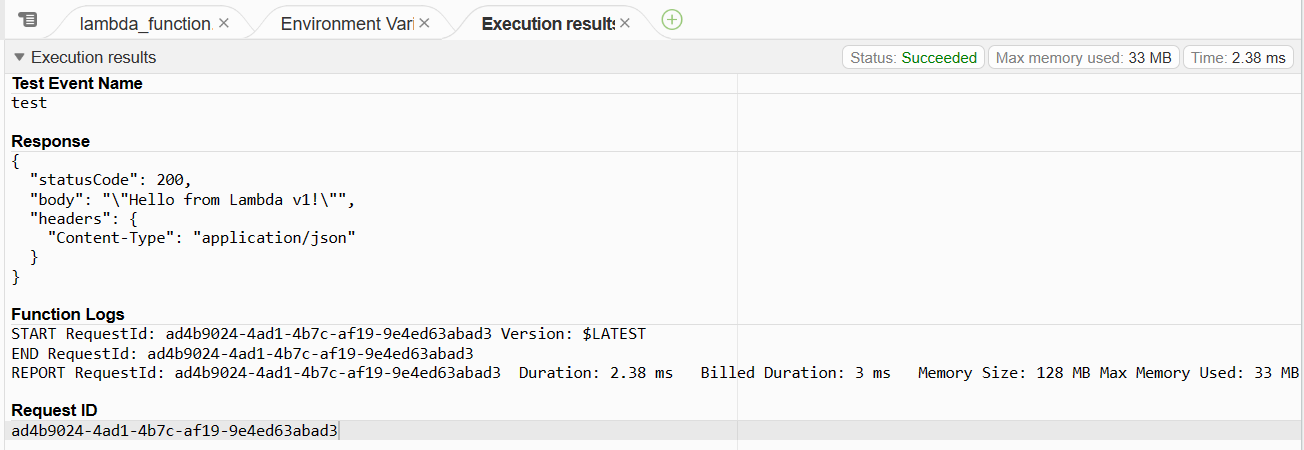
## API Gateway Stages and Deployments Hands-on:

## Stage Variables

### 1. Lambda Function Versions and Aliases

1. **Create a Lambda Function:**
   * Name: api-gateway-stage-variables-get
   * Runtime: Python 3.11
   * Code:

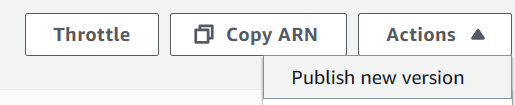


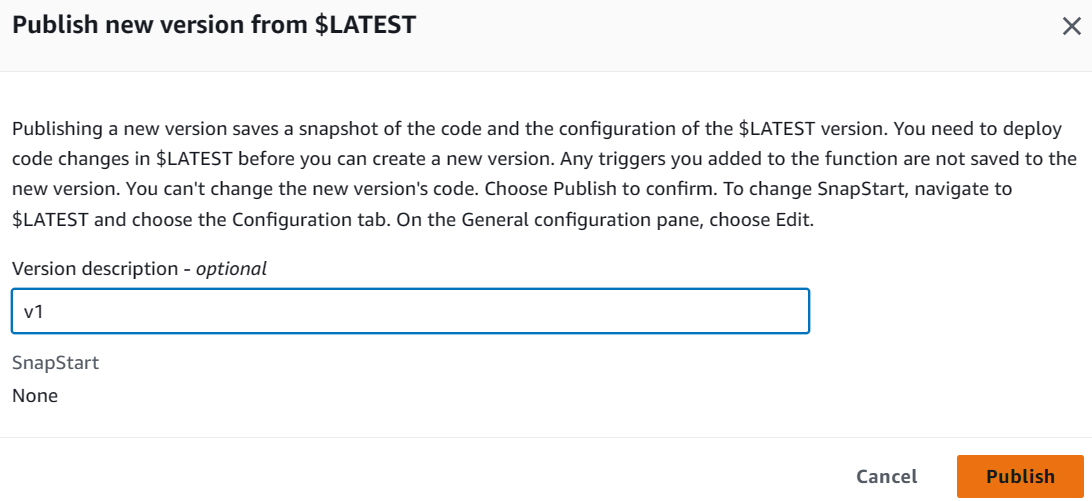
****

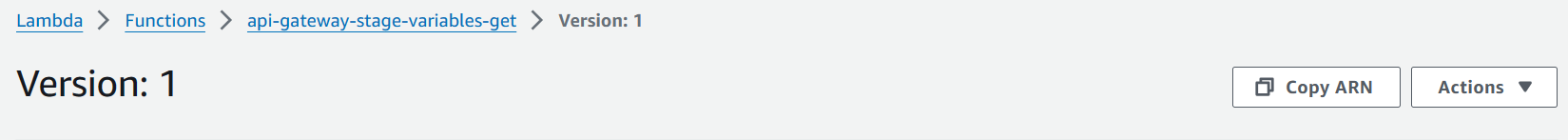
1. **Create Versions:**

**Action > publish new version**

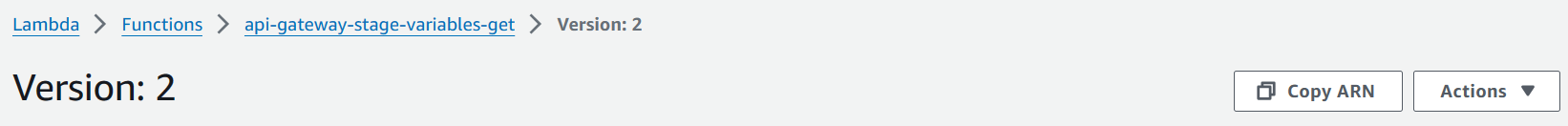
Version 1: Code - "Hello from Lambda v1" (Deploy and Test)







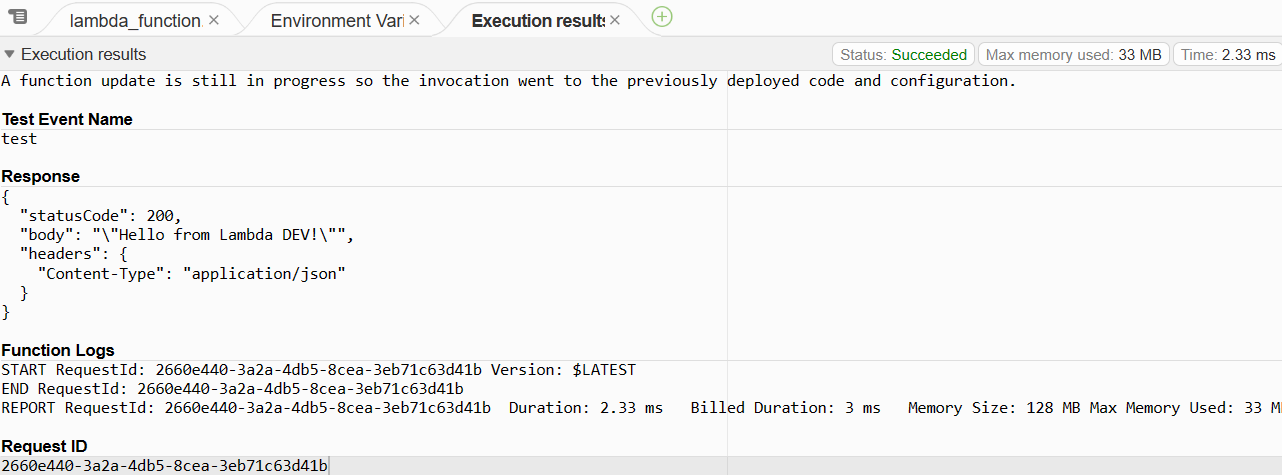
Version2: Update your function code “Hello from lambda v2”. Deploy and test it. Publish v2.



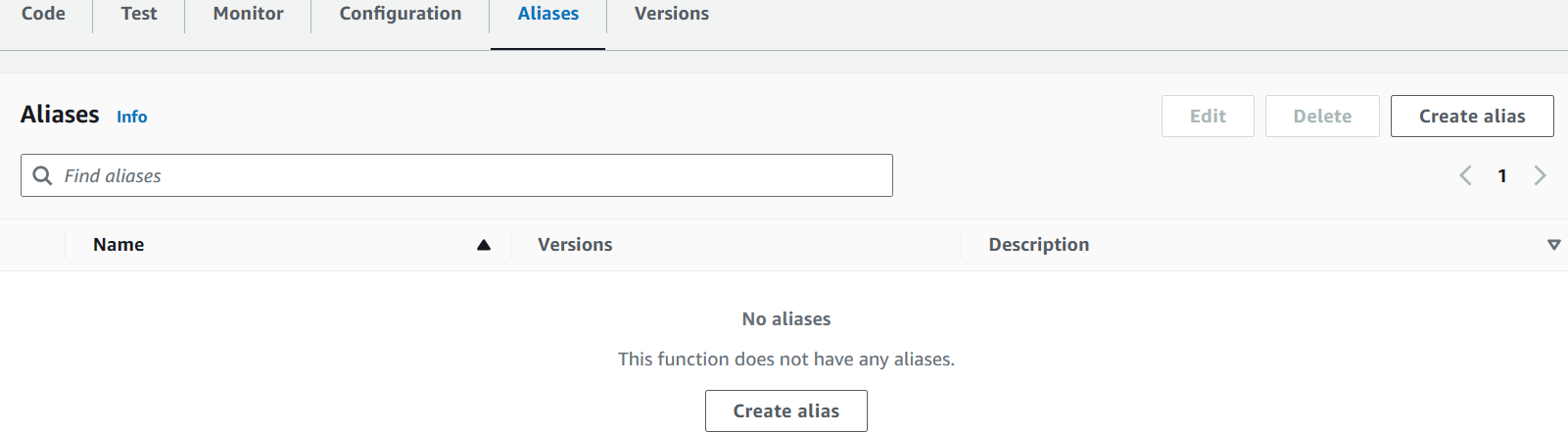
Update your function code “Hello from lambda Dev”

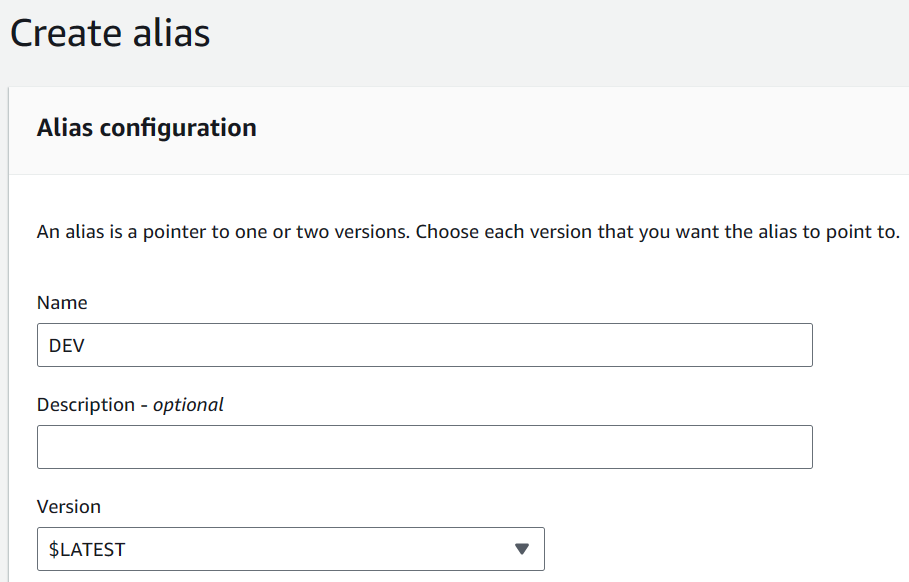
(Deploy and Test, **Don't Publish**)

**Note:** We won't publish version 3, keeping it as the latest for the DEV alias.



1. **Create Aliases:**

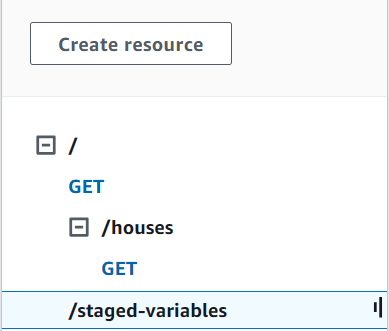




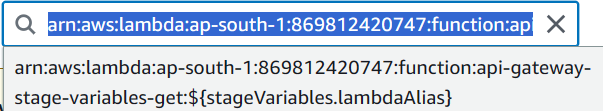
* + DEV: Points to $LATEST (Latest Version)
  + TEST: Points to Version 2
  + PROD: Points to Version 1

### **2. API Gateway Integration with Stage Variables**

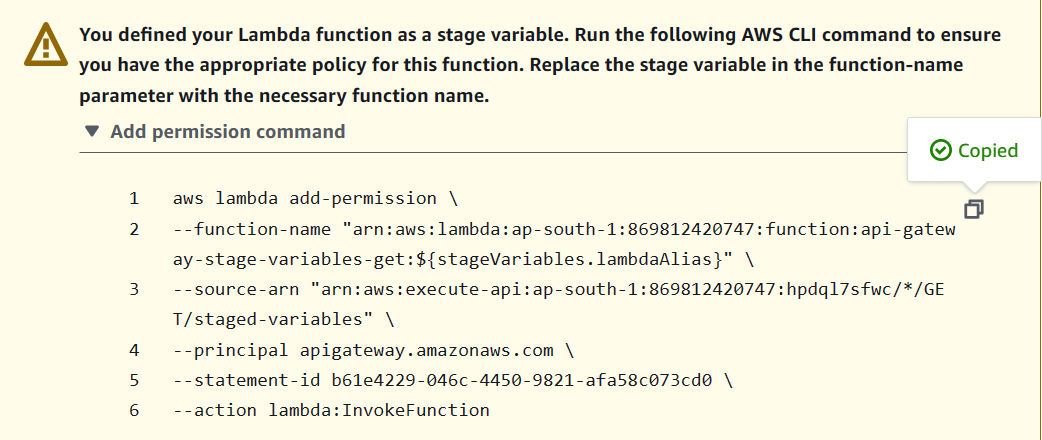
1. **Create a Resource:** Name: staged-variables



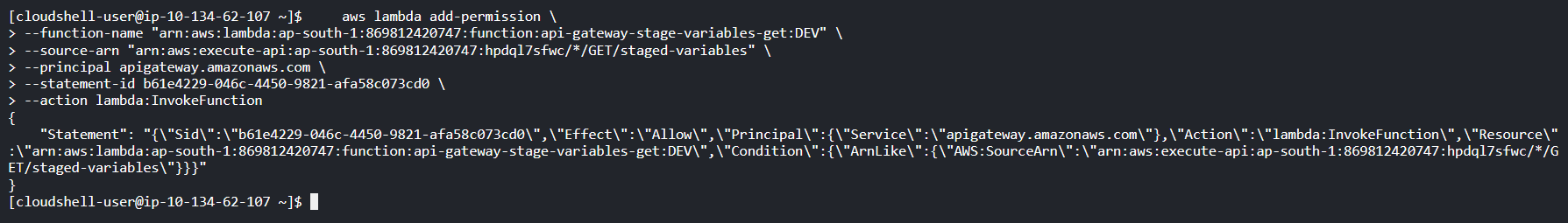
1. **Create a GET Method:**
   * Integration Type: Lambda Function
   * Lambda Function: Choose Proxy Integration
   * URI: arn:aws:lambda:{region}:{account-id}:function:{lambda-function-name}:${stageVariables.lambdaAlias}



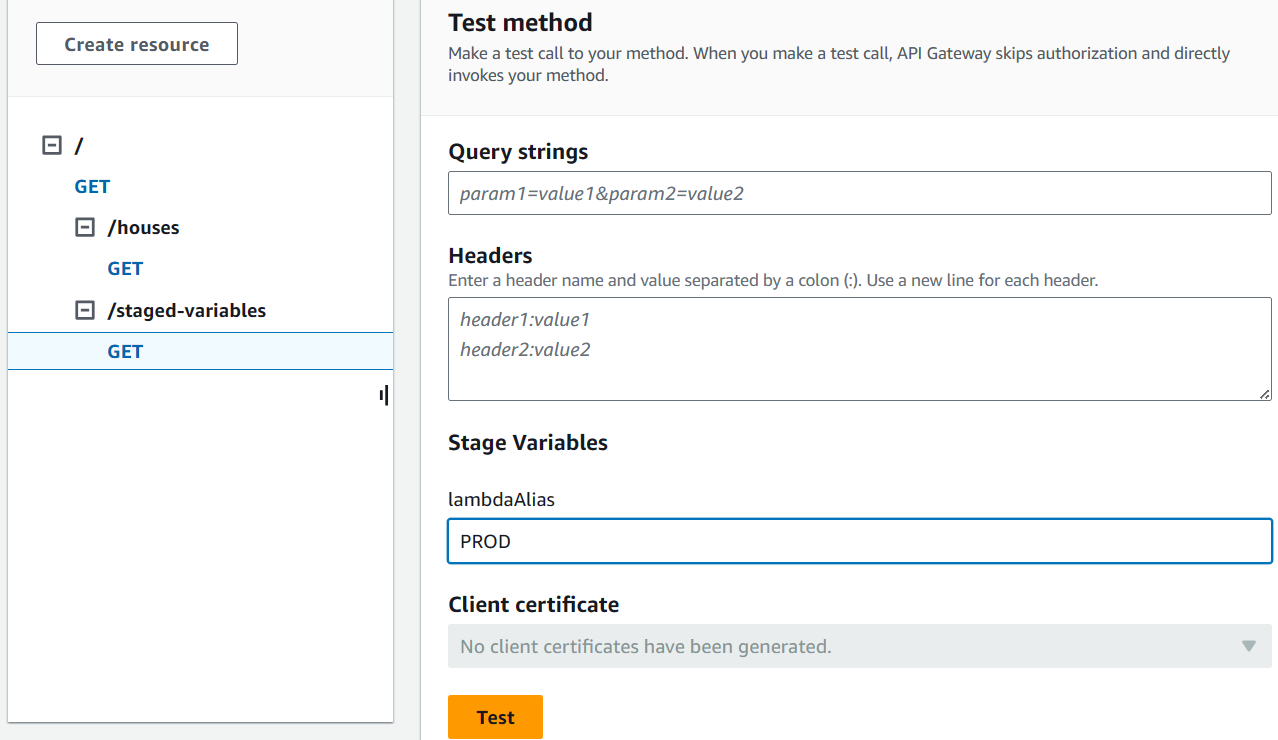
1. **Set Permissions:**



* + Copy the permission command provided in the guide.
  + **Fix the Bug:** Remove the duplicated ARN and the trailing dollar sign.
  + Modify the command to create separate permissions for DEV, TEST, and PROD aliases allowing API Gateway to invoke them.
  + Run the commands in CloudShell.

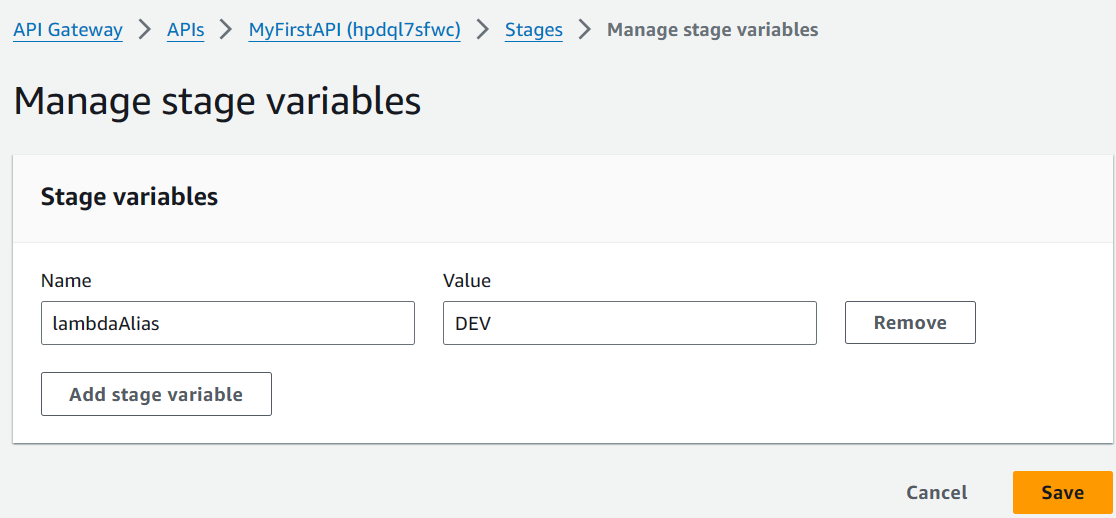


1. **Test with Stage Variables:**
   * In the Test UI, under "Test for stage variables"
     + Enter the desired alias (e.g., PROD) to invoke the corresponding Lambda version.
   * Verify successful responses with "Hello from lambda v#" based on the invoked alias.

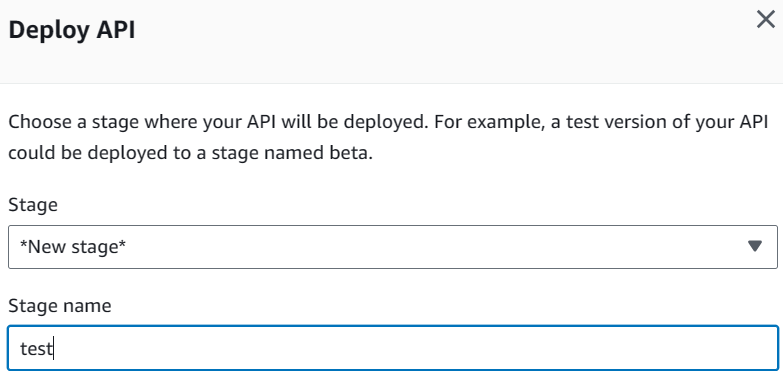


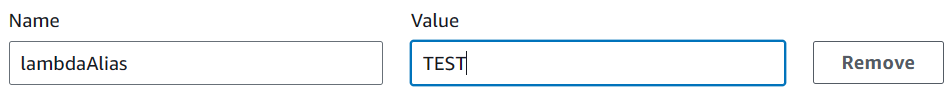
### 3. Deploying API with Stage Variables

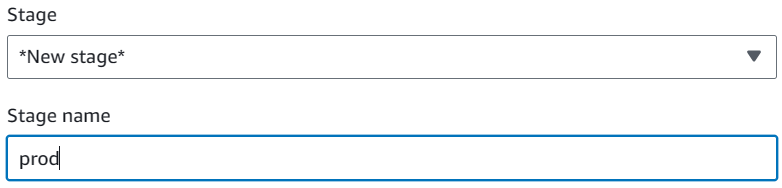
1. **Deploy API to DEV Stage:**
   * In the Stage Editor for DEV, set a new stage variable:
     + Name: lambdaAlias
     + Value: DEV

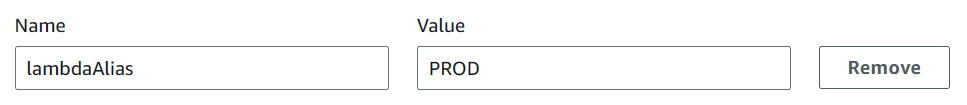


1. **Deploy API to TEST and PROD Stages:**
   * Repeat step 1 for TEST and PROD stages, setting the lambdaAlias variable to TEST and PROD respectively.

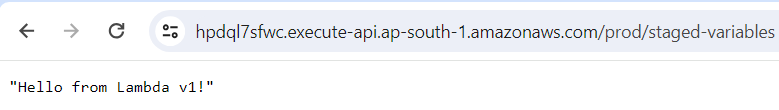








1. **Test Deployed API:**
   * Use the deployed API URL with different stages (/stage-variables):



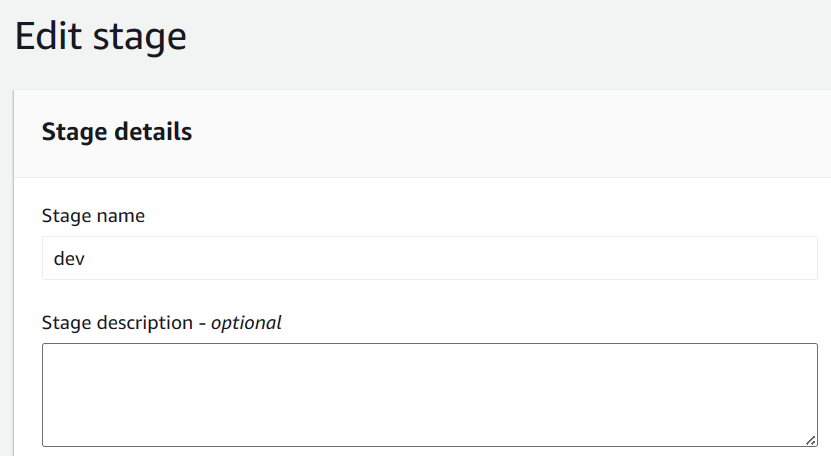
* + - Main URL: Invokes PROD alias (Hello from lambda v1)
    - URL with /dev: Invokes DEV alias (Hello from Lambda DEV)
    - URL with /test: Invokes TEST alias (Hello from Lambda v2)

This demonstrates how stage variables allow you to control which Lambda version (through aliases) your API invokes based on the deployment stage.

**API Gateway Stage Configuration Options**

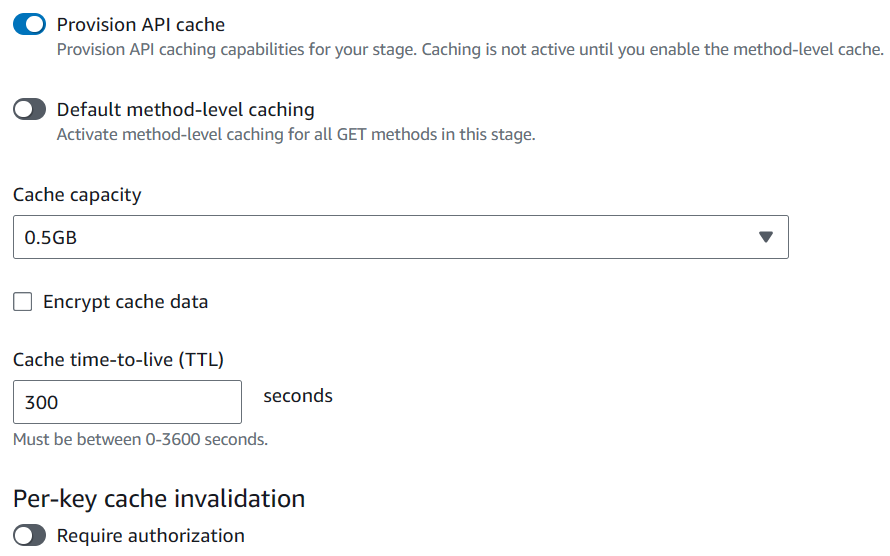
**General Settings:**

* **Stage Description:** Provide a descriptive name for your stage (e.g., "Development," "Staging," "Production").

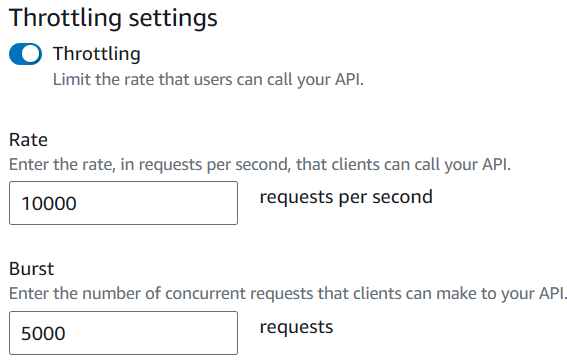


**Performance Optimization:**

* **API Caching:** Improve response times by caching frequently accessed API responses. Configure cache invalidation timeouts as needed.

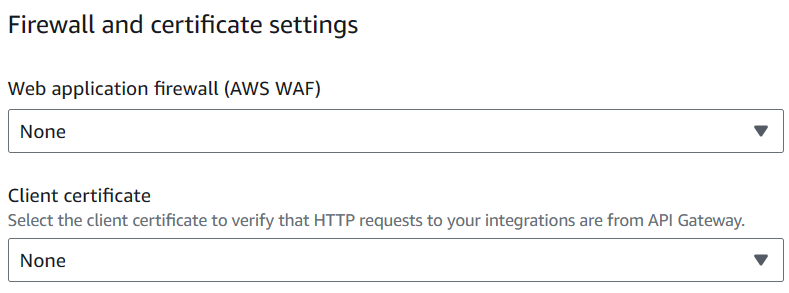


* **Throttling:** Limit the number of requests your API can receive per second (rate limit) and allow for burst requests to handle sudden spikes.



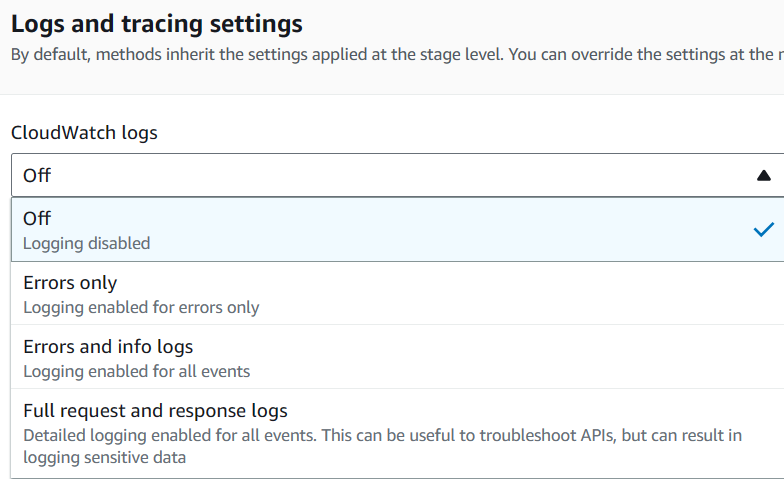
**Security:**

* **Firewall Configuration:** Employ AWS WAF (Web Application Firewall) to protect your API from malicious attacks.
* **Client Certificate Verification:** Require clients to present valid certificates for API access, enhancing security.

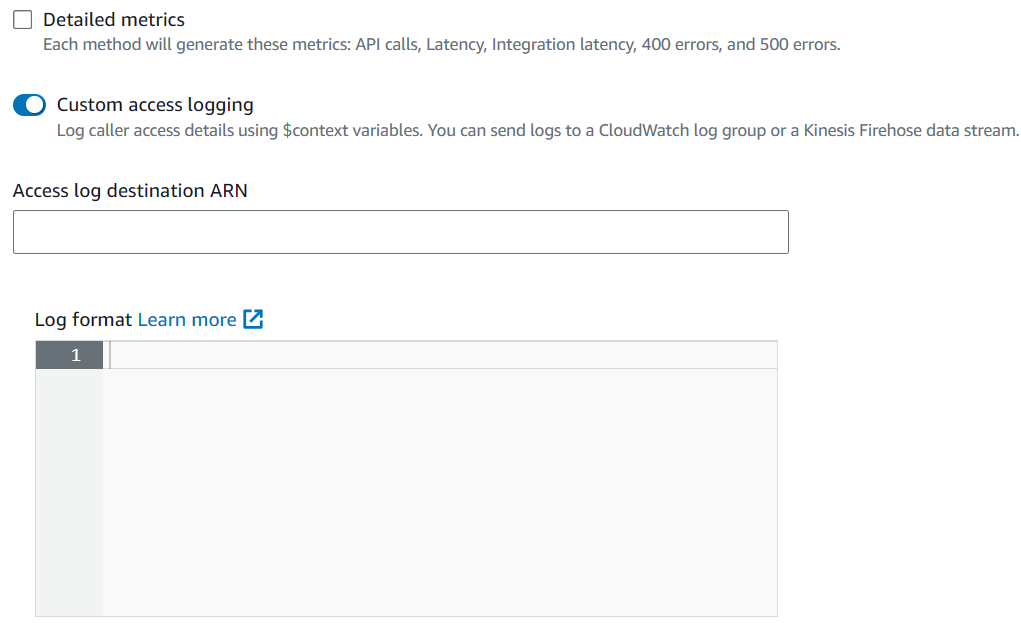


**Monitoring and Logging:**

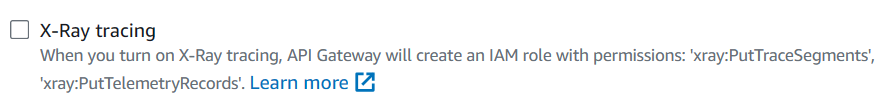
* **Logging and Tracing:** Set up CloudWatch Logs to capture API request/response logs and trace details for debugging purposes.
  + **Log Levels:** Choose the logging level (Error, Info, etc.) based on your needs.



* + **Custom Access Logging:** Define custom logging formats for capturing specific data.

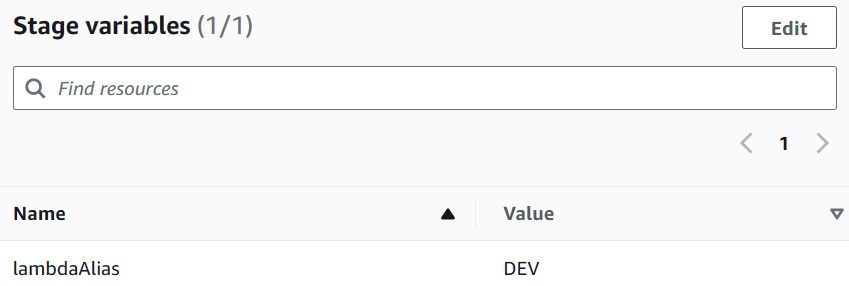


* **X-Ray Tracing:** Enable X-Ray integration for detailed tracing across your API and other AWS services.



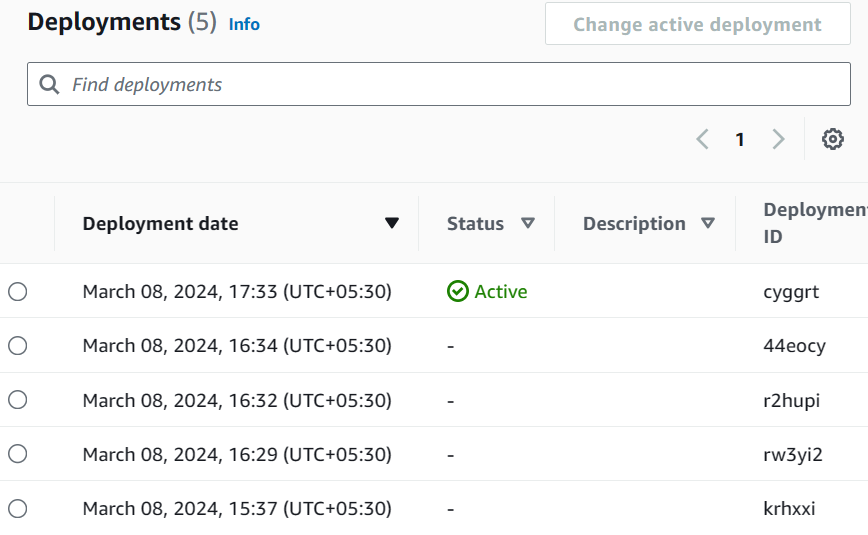
**Stage Variables:**

* Dynamically configure your API behaviour by defining stage-specific variables accessible within your Lambda functions.



**Deployment and Versioning:**

* **Deployment History:** Track a record of all deployments made to your stage.



* **Documentation History:** Maintain a history of any documentation associated with the stage.
* **Stage Tags:** Assign tags to your stage for easier categorization and resource management.