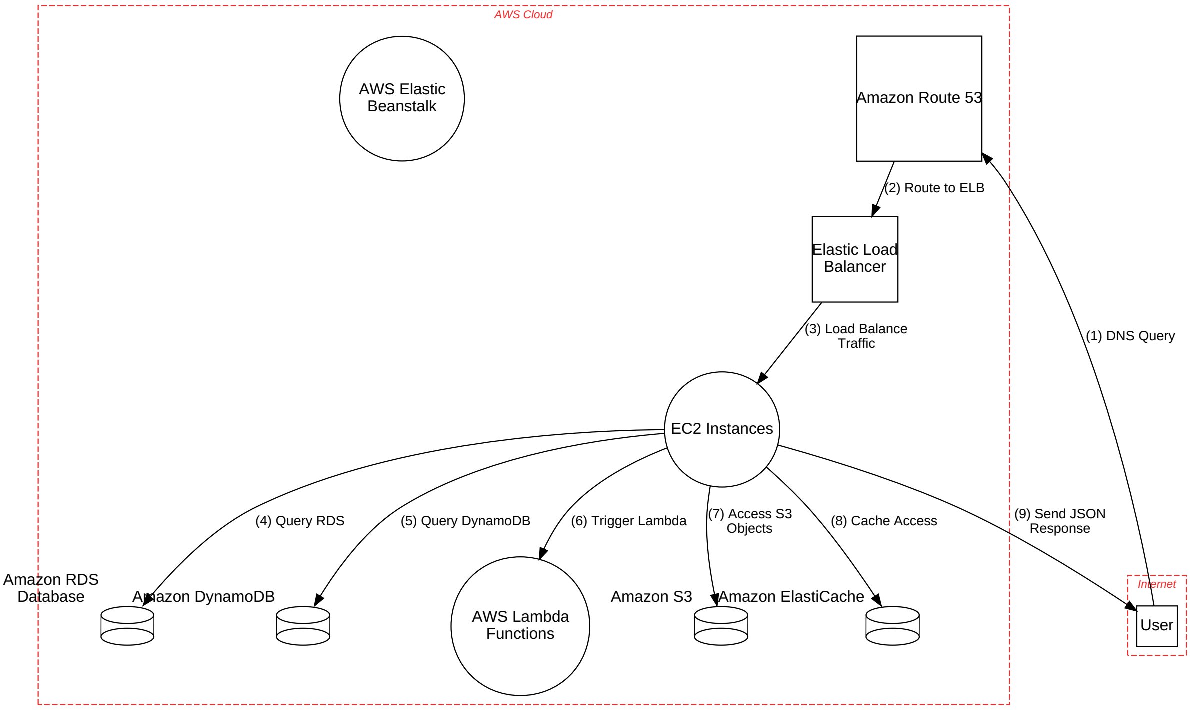
This is an auto-generated Data Flow Diagram, assembled by GPT-4 Threat Modeling Agents. The system reviews the specified application architecture, and generates a data flow diagram using pytm and GraphViz. The result may still contain errors.



Appendix

Original Prompt and Inputted App Architecture

Create a data flow diagram for the following app architecture: In a web application hosted on AWS, the data flow begins with the user's interaction with the front-end, which triggers an HTTP request. This request is routed through Amazon Route 53 to an Elastic Load Balancer, which then directs the traffic to the appropriate EC2 instances where the application is hosted. The application code, running on an AWS Elastic Beanstalk environment, processes the request, which includes querying an Amazon RDS database and Amazon DynamoDB table to retrieve or store data. AWS Lambda functions are also utilized for serverless computation. The application interacts with additional AWS services like S3 for object storage, and Amazon ElastiCache to access frequently requested data quickly. Once the server-side processing is complete, the data is formatted (as JSON) and sent back through the Internet to the user's browser, where it is rendered, and any dynamic client-side actions are handled by JavaScript.

Usage Costs

```
#### Total Cost: $0.04347 USD
#### Input Tokens Cost: $0.0 USD
#### Output Tokens Cost: $0.04347 USD
```

Conversation Log

| STARTING CONVERSATION: Create a data flow diagram for the following app architecture: In a web application hosted on AWS, the data flow begins with the user's interaction with the front-end, which triggers an HTTP request. This request is routed through Amazon Route 53 to an Elastic Load Balancer, which then directs the traffic to the appropriate EC2 instances where the application is hosted. The application code, running on an AWS Elastic Beanstalk environment, processes the request, which includes querying an Amazon RDS database and Amazon DynamoDB table to retrieve or store data. AWS Lambda functions are also utilized for serverless computation. The application interacts with additional AWS services like S3 for object storage, and Amazon ElastiCache to access frequently requested data quickly. Once the server-side processing is complete, the data is formatted (as JSON) and sent back through the Internet to the user's browser, where it is rendered, and any dynamic client-side actions are handled by JavaScript.DESCRIPTION_END | user_proxy (to chatbot): | | Create a data flow diagram for the following app architecture: In a web application hosted on AWS, the data flow begins with the user's interaction with the front-end, which triggers an HTTP request. This request is routed through Amazon Route 53 to an Elastic Load Balancer, which then directs the traffic to the appropriate EC2 instances where the application is hosted. The application code, running on an AWS Elastic Beanstalk environment, processes the request, which includes querying an Amazon RDS database and Amazon DynamoDB table to retrieve or store data. AWS Lambda functions are also utilized for serverless computation. The application interacts with additional AWS services like S3 for object storage, and Amazon ElastiCache to access frequently requested data quickly. Once the server-side processing is complete, the data is formatted (as JSON) and sent back through the

Internet to the user's browser, where it is rendered, and any dynamic client-side actions are handled by JavaScript.DESCRIPTION_END | | -- | chatbot (to user_proxy): | | Here is the pytm script for the described web application hosted on AWS: | | `python | tm = TM("Web Application Data Flow") | tm.description = "Threat model for a web application hosted on AWS with various components." | tm.isOrdered = True | tm.mergeResponses = True | | internet = Boundary("Internet") | | aws_cloud = Boundary("AWS Cloud") | | user = Actor("User") | user.inBoundary = internet | | route53 = ExternalEntity("Amazon Route 53") | route53.inBoundary = aws_cloud | | elb = ExternalEntity("Elastic Load Balancer") | elb.inBoundary = aws_cloud | | ec2 = Process("EC2 Instances") | ec2.inBoundary = aws_cloud | | elastic_beanstalk = Process("AWS Elastic Beanstalk") | elastic_beanstalk.inBoundary = aws_cloud | | rds = Datastore("Amazon RDS Database") | rds.inBoundary = aws_cloud | | dynamodb = Datastore("Amazon DynamoDB") | dynamodb.inBoundary = aws_cloud | | lambda_function = Process("AWS Lambda Functions") | lambda_function.inBoundary = aws_cloud | | s3 = Datastore("Amazon S3") | s3.inBoundary = aws_cloud | | elasticache = Datastore("Amazon ElastiCache") | elasticache.inBoundary = aws_cloud | | js = Data("Dynamic JS Actions") | json_data = Data("JSON Data") | | # Dataflows | user_to_route53 = Dataflow(user, route53, "DNS Query") | | route53_to_elb = Dataflow(route53, elb, "Route to ELB") | | elb_to_ec2 = Dataflow(elb, ec2, "Load Balance Traffic") | | ec2_to_rds = Dataflow(ec2, rds, "Query RDS") | | ec2_to_dynamodb = Dataflow(ec2, dynamodb, "Query DynamoDB") | | ec2_to_lambda = Dataflow(ec2, lambda_function, "Trigger Lambda") | | ec2_to_s3 = Dataflow(ec2, s3, "Access S3 Objects") | | ec2_to_elasticache = Dataflow(ec2, elasticache, "Cache Access") | | ec2_to_user = Dataflow(ec2, user, "Send JSON Response") |` | | Now we'll use this script to generate the data flow diagram. | * **Suggested function Call: python |** ***Arguments: | {"cell":"tm = TM(\"Web Application Data Flow\")\ntm.description = \"Threat model for a web application hosted on AWS with various components.\"\ntm.isOrdered = True\ntm.mergeResponses = True\n\ninternet = Boundary(\"Internet\")\n\naws_cloud = Boundary(\"AWS Cloud\")\n\nuser = Actor(\"User\")\nuser.inBoundary = internet\n\nroute53 = ExternalEntity(\"Amazon Route 53\")\nroute53.inBoundary = aws_cloud\n\nelb = ExternalEntity(\"Elastic Load Balancer\")\nelb.inBoundary = aws_cloud\n\nec2 = Process(\"EC2 Instances\")\nec2.inBoundary = aws_cloud\n\nelastic_beanstalk = Process(\"AWS Elastic Beanstalk\")\nelastic_beanstalk.inBoundary = aws_cloud\n\nrds = Datastore(\"Amazon RDS Database\")\nrds.inBoundary = aws_cloud\n\ndynamodb = Datastore(\"Amazon DynamoDB\")\ndynamodb.inBoundary = aws_cloud\n\nlambda_function = Process(\"AWS Lambda Functions\")\nlambda_function.inBoundary = aws_cloud\n\ns3 = Datastore(\"Amazon S3\")\ns3.inBoundary = aws_cloud\n\nelasticache = Datastore(\"Amazon ElastiCache\")\nelasticache.inBoundary = aws_cloud\n\njs = Data(\"Dynamic JS Actions\")\njson_data = Data(\"JSON Data\")\n\n# Dataflows\nuser_to_route53 = Dataflow(user, route53, \"DNS Query\")\n\nroute53_to_elb = Dataflow(route53, elb, \"Route to ELB\")\n\nelb_to_ec2 = Dataflow(elb, ec2, \"Load Balance Traffic\")\n\nec2_to_rds = Dataflow(ec2, rds, \"Query RDS\")\n\nec2_to_dynamodb = Dataflow(ec2, dynamodb, \"Query DynamoDB\")\n\nec2_to_lambda = Dataflow(ec2, lambda_function, \"Trigger Lambda\")\n\nec2_to_s3 = Dataflow(ec2, s3, \"Access S3 Objects\")\n\nec2_to_elasticache = Dataflow(ec2, elasticache, \"Cache Access\")\n\nec2_to_user = Dataflow(ec2, user, \"Send JSON Response\")"} |*** ****** | | -- | | >>>>>>>> EXECUTING FUNCTION python... |