

SRV313

AWS re:INVENT

Building Resilient, Multi-Region Serverless Applications

Magnus Bjorkman, AWS Solution Architect

Stefano Buliani, AWS Specialist Solution Architect

November 28, 2017

AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Serverless computing already provides high availability and fault tolerance for your application by default. However, you can build serverless applications that are deployed across multiple regions in order to further increase your availability and fault tolerance. In this session, we show you how to architect a multi-region serverless application with Amazon API Gateway and AWS Lambda that can route end users to the appropriate region to achieve optimal latency or availability. Learn about the different options for running an active/active versus an active/passive multi-region setup, and the setup for failing over between regions.

What to expect from the session

- “Food-for-Thought” – a real world serverless application
- Why go multi-region?
- Decision tree – do even need to go multi-region?
- One service at a time – routing and data replication concepts:
 - Browsing catalog
 - Updating catalog
 - Receiving orders
- Developer tools and deployments
- Summary

We have divided the application into 3 main microservices, the Browsing catalog, Updating catalog, and Receiving orders microservices.

Example application

"Food-for-Thought"

- Order food from any restaurant near you and get it delivered at your home
 - Store and browse restaurant menu items
 - Accept orders and process payments
- Company doing business through-out North America and Europe
 - Customers should see low latency and high performance on either continent.
- Very high peak to average ratio of traffic around meal times
 - A highly fault-tolerant system is critical as reputation loss and revenue loss would be significant for an outage during those hours

AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Photo by [Brigitte Tohm](#) on [Unsplash](#)



Benefits of AWS Lambda and serverless



No servers to provision
or manage



Never pay for idle



Scales with usage



Availability and fault
tolerance built in

AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



This is a very good use case for Serverless

Why is "Food-for-Thought" going multi-region?



Data sovereignty



AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



We want to place the application in a region that is closer to the customer so that they get the improved latency and performance, also to have the application in a HA manner by using more than one region.

What "Food-for-Thought" should consider when going multi-region

"Food-for-Thought" – three key services



Customer



Browse Catalog

- View Restaurants
- View Menus



Update Catalog

- Add, Update and Remove Restaurants
- Add, Update and Remove Menus



Submit Order

- Order Processing
- Payment Processing
- Order Confirmation



Administrator

AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Does it need to be multi-region?



- Can we do business **without this service**? How much redundancy do we need?

- How does this service access **persistent storage**?

AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



How much will this cost?

- Lambda and Amazon API Gateway only charge you for your code actually runs – **traffic is the same, price is the same, doesn't matter how many AWS Regions**
- Database and file storage will incur additional charges; **for fully redundant services, double your costs**
- Data transfer out to internet costs can **vary depending on the AWS Region**

Monitoring and logging

- Monitoring
 - Fine-grained monitoring within the AWS Region
 - Enough monitoring from outside AWS Region to detect issue in an AWS Region, especially with monitoring system
- Logging
 - Consider replicating logs if they are critical to operations

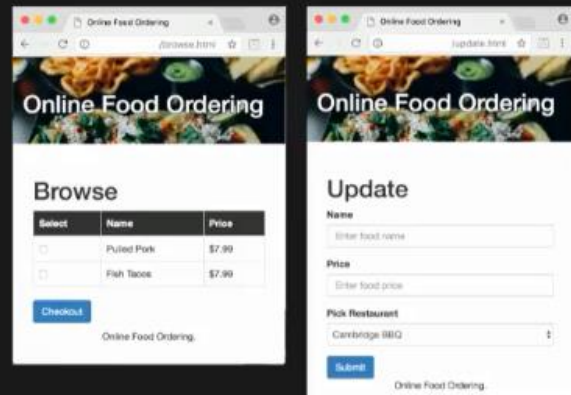
Data sovereignty

- For data under purview of data sovereignty laws:
 - Implement requirements into application code and do not solely rely on routing features like geographic routing in Amazon Route 53
 - Consider using home AWS Regions for users and make sure that the code/data is always executed/stored in that particular AWS Region for that particular user.
- Use separate Amazon Cognito user pools in each region

Browsing and updating catalog

Browsing and updating catalog

- Availability:
 - It is critical to be able to browse the catalog so customers can create orders.
 - It is less critical to be able to update the catalog.
- Consistency:
 - It is required to have at least eventual read consistency.
- External Dependencies:
 - None



The image displays two side-by-side screenshots of a web application titled "Online Food Ordering".

The left screenshot shows the "Browse" page. It features a header with the application name and a hero image of food. Below the header is a section titled "Browse" containing a table with two columns: "Select" and "Name". The table lists two items: "Pulled Pork" with a price of "\$7.99" and "Fish Tacos" with a price of "\$7.99". There is a "Checkout" button at the bottom of the table.

The right screenshot shows the "Update" page. It also has the same header and hero image. Below the header is a section titled "Update" with three input fields: "Name" (with placeholder text "Enter food name"), "Price" (with placeholder text "Enter food price"), and "Pick Restaurant" (a dropdown menu showing "Cambridge BBQ"). A "Submit" button is at the bottom.

Browsing the catalog



- Is this service **truly core to our business**?
 - **YES** – we need the catalog in order to accept orders
- How does this service access **persistent storage**?
 - Data replicated to all AWS Regions from master copy

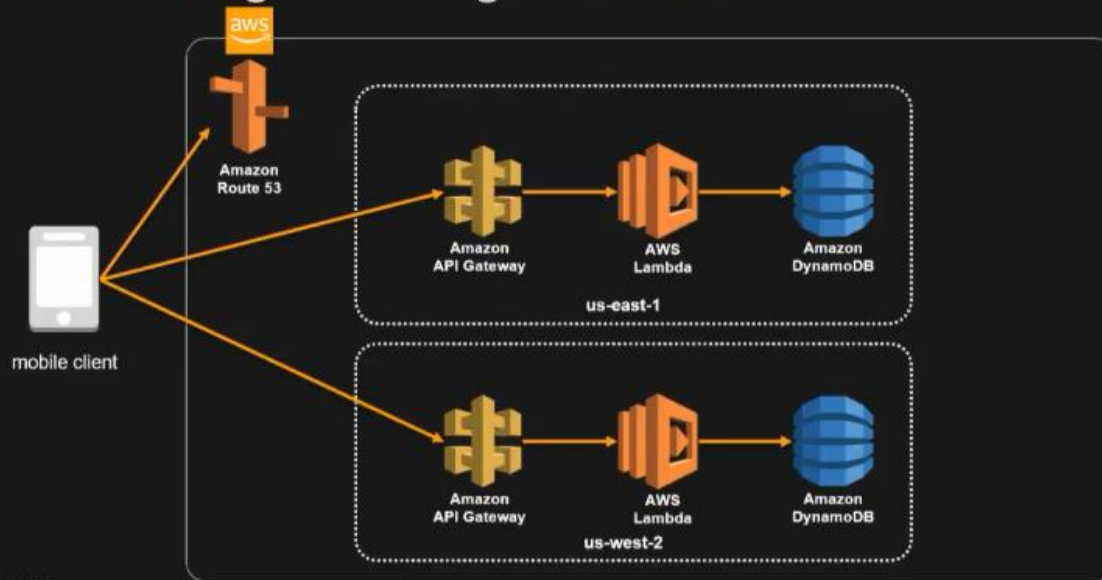
We will use multi-regions and use DNS routing to direct the customer to the closest region to them. We are going to use near real-time replication because of the need for the smallest latency

Updating the catalog



- Is this service **truly core to our business**?
 - **NO** – even if the master AWS Region is down we can still accept orders.
 - The architectural complexity of master/master replication is not worth the return.

Browsing catalog - architecture



aws re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Browsing catalog – architecture highlights

- Active-Active multi-regional API
- Deploy same stack to multiple AWS Regions
- Read-Only; access Amazon DynamoDB tables in local AWS Region
- Optional: For public and content more static in nature, put Amazon CloudFront in front of API.

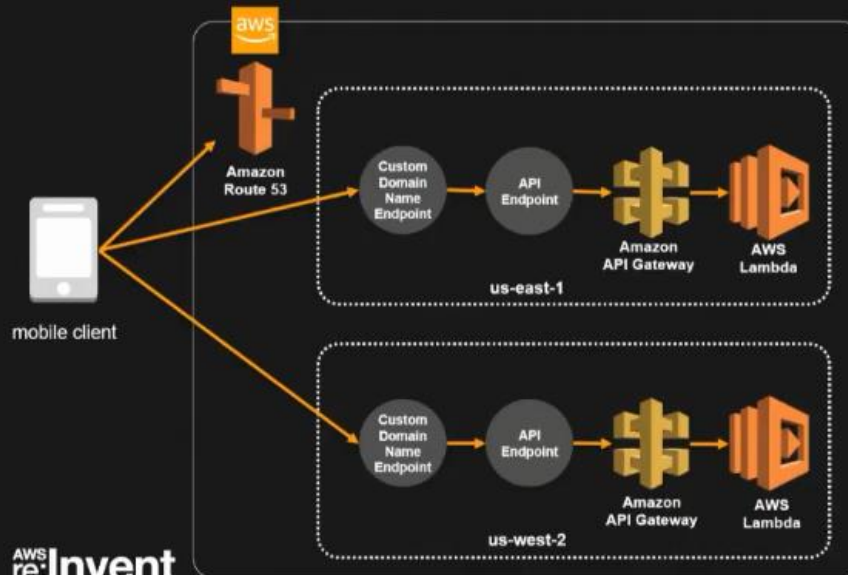
The Active-Active multi-regional API means that both API Gateways are taking live traffic at the same time.

API Gateway: Regional endpoints

- Endpoint configuration types
 - Edge optimized
 - Regional
- Endpoint configuration types are applicable for both custom domain names as well as APIs
- Regional endpoints: custom domains are unique per AWS Region

A custom domain name is like *api.foodforthought.com* that we can now use in both region.

Multi-region API Gateway setup



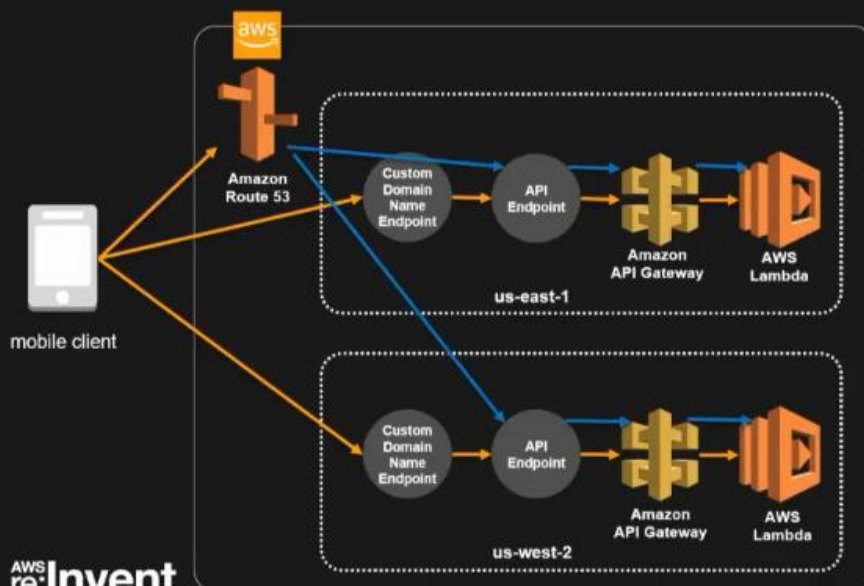
- Endpoints in AWS Regions
- Amazon Route 53 Routing (e.g. Latency, Geolocation)
- Multi-region backends
- Active-Active



aws re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Failover across AWS Regions with API Gateway



- Amazon Route 53 health checks
- Provide a path that does a deep ping

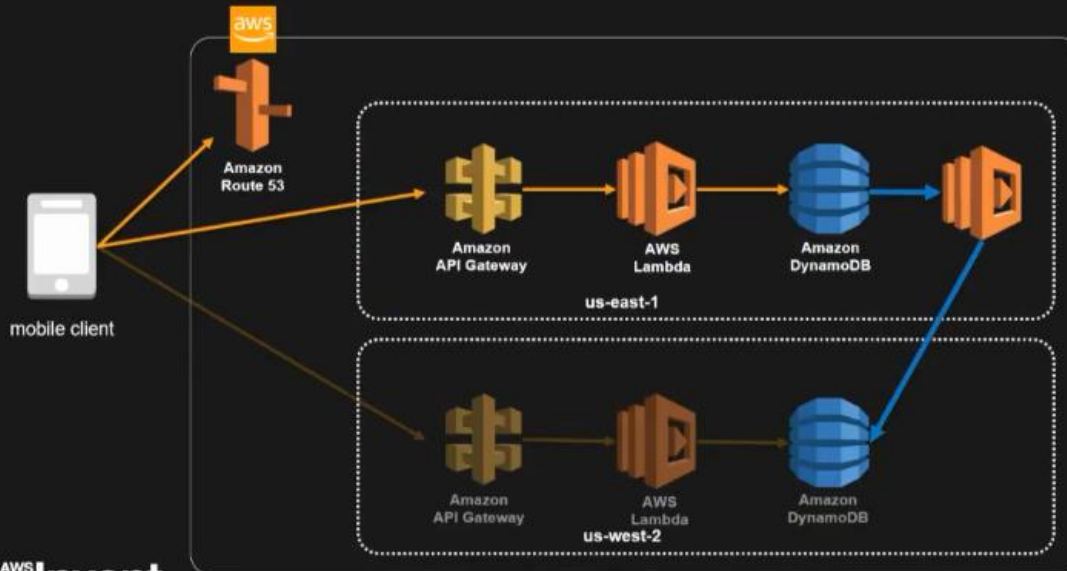


aws re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

We need to provide paths that the health check can call along the paths for making sure the API is working

Updating catalog - architecture

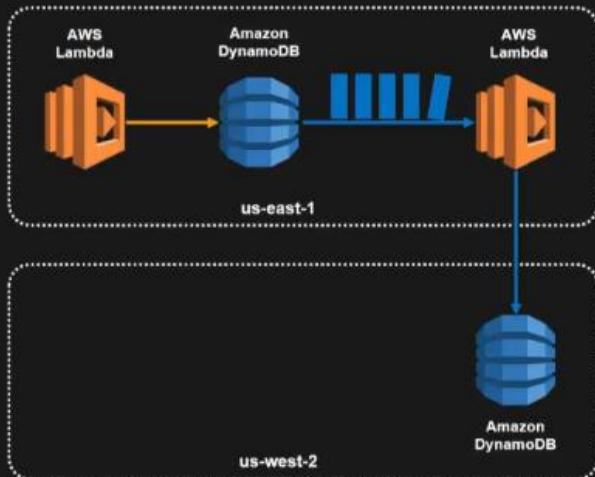


This is a single region setup with us using a lambda for replicating updates and entries to the top DynamoDB tables into the backup table.

Updating catalog – architecture highlights

- Single-region API. Updates are lower criticality than reads. Graceful degradation.
- Data availability is critical for “Browse Catalog” use case. Replicate to other AWS Regions using Amazon DynamoDB Streams.
 - Simplify
 - No data conflicts
 - No split brain

Master/replica replication with Lambda



```
for record in event['Records']:
    print(record['eventID'])
    print(record['eventName'])
    print("DynamoDB Record: "
          + json.dumps(record['dynamodb'], indent=2))

    if (record['eventName'] == 'INSERT') or
       (record['eventName'] == 'MODIFY'):
        client.put_item(Table=resource_properties['TargetDDB'],
                        Item=record['dynamodb']['NewImage'])
    elif record['eventName'] == 'REMOVE':
        client.delete_item(Table=resource_properties['TargetDDB'],
                           Key=record['dynamodb']['Keys'])
```

We are going to use DynamoDB Streams to do the replication to the Lambda function as the event destination that would do the insertion/updating of the backup DynamoDB tables in the other region and replay all the updates.

Submit order

Submit order

- Availability:
 - It is critical to be able to write new orders.
 - It is critical to be able to read and update existing orders.
- Consistency:
 - It is required to have at least eventual read consistency for status of existing orders.
 - It is required to have strong consistency for read-before-write.
- External Dependencies:
 - Payment System



Decision tree



AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



- Is this service **truly core to our business**?
 - **YES** – it's our primary revenue source
- How does this service access **persistent storage**?
 - Each order has a master AWS Region. Orders are replicated but can only be modified in their master AWS Region.

Submit order – almost Master / Master

- True master/master replication is very hard:
 - Multi-region ACID transactions are impractical
- Orders have a master AWS Region
- Writes always go to the master AWS Region
- Reads for processing can happen in any AWS Region
- Client logic to ensure updates always go to the master AWS Region
- Other AWS Regions should refuse to modify the order if they receive a request

Why home AWS Region if multi-master is possible

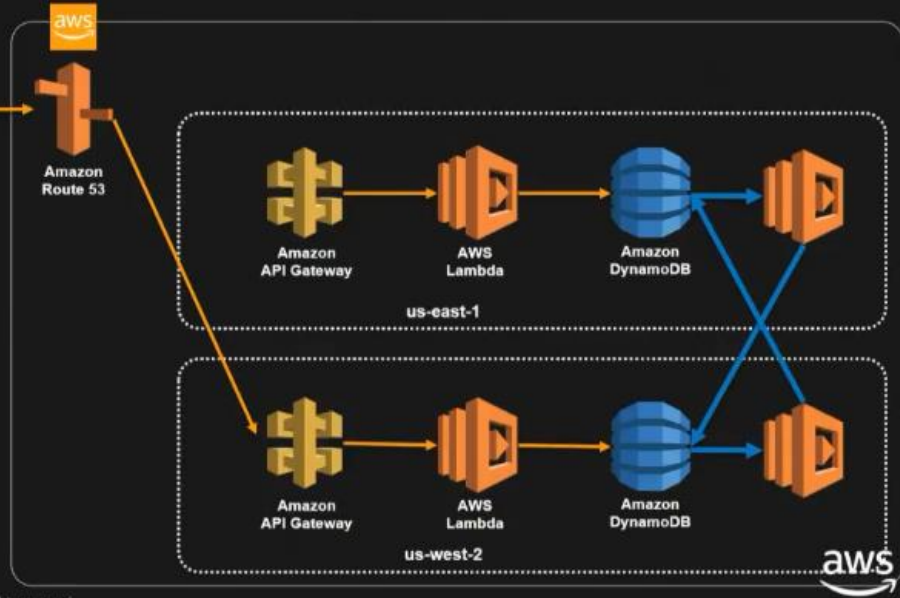
- Multi-master is very complex – chances of bugs are higher
- Eventual consistency in reads can create a frustrating customer experience – I don't know what the true state of my order is
 - Erodes customer trust quickly
- Losing an entire AWS Region is highly unlikely, much less likely than order state being inconsistent across the two AWS Regions

Application state – client side

1. Client uses the DNS-balanced endpoint to create an order
2. AWS Region creates the order and sets itself as the master

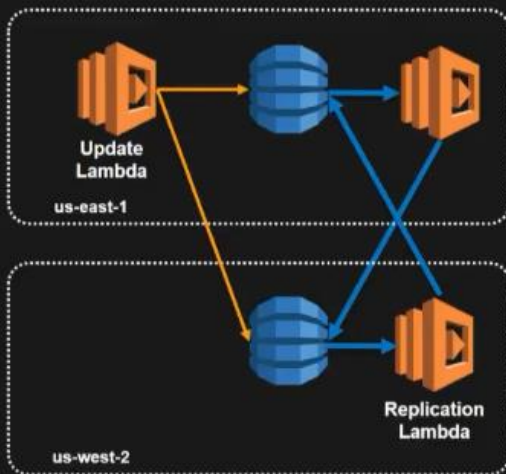
AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



We need to use additional lambdas for syncing the tables together as above.

Home AWS Region replication



orderId	homeRegion	userId
1234	us-east-1	9012
5678	us-west-2	3456

- Each order/record has a home AWS Region. Only write to home AWS Region.
- Read from home AWS Region unless not available.
- Mark initial record for replication to avoid circular updates.

Once we create an order we will send back an **orderId** for it with the **homeRegion** and the **userId** values. This enables our backends to determine if they are the home region or not

Application state – client side

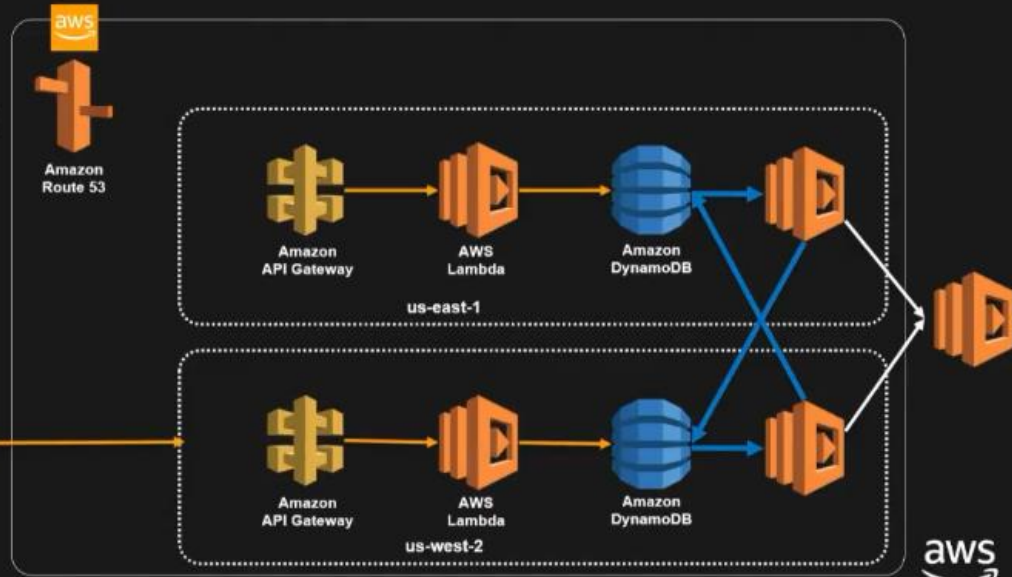
3. Client addresses home AWS Region Directly to update order

4. Once written, any AWS Region can process the order



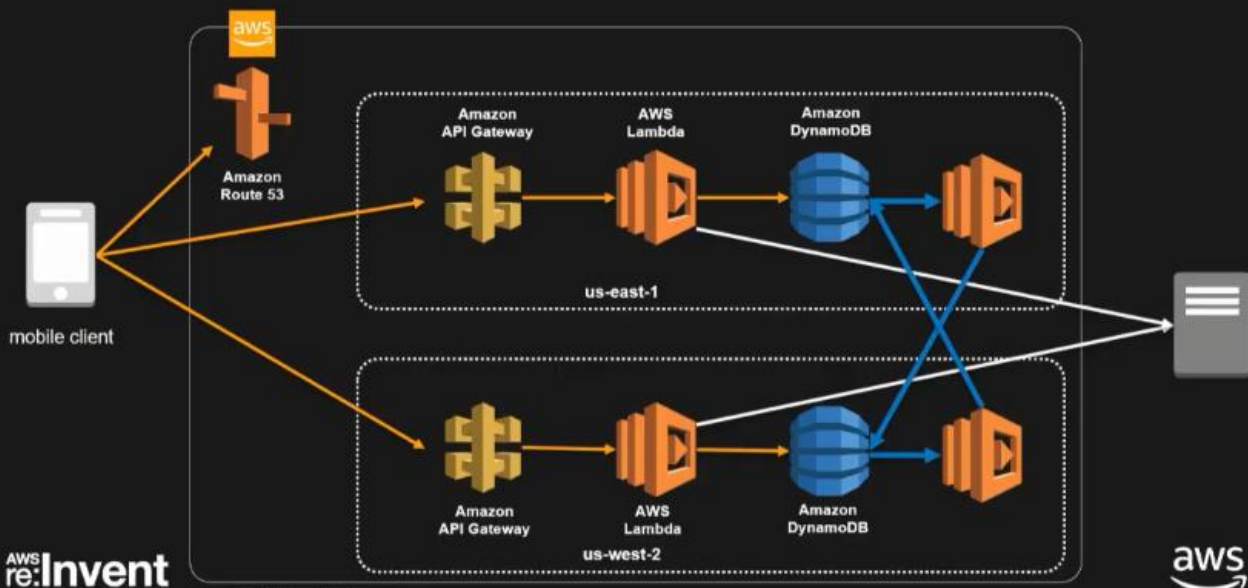
aws re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



We now have 3 endpoints, a global endpoint and 2 region specific aliases. The order edit command is sent to the region where the order was created initially, we are also replicating the changes to the other regions for quick read capability from any region.

Submit order - architecture



There is a 3rd party payment processing system that we need to use when an order is submitted.

Master/Master replication: other options

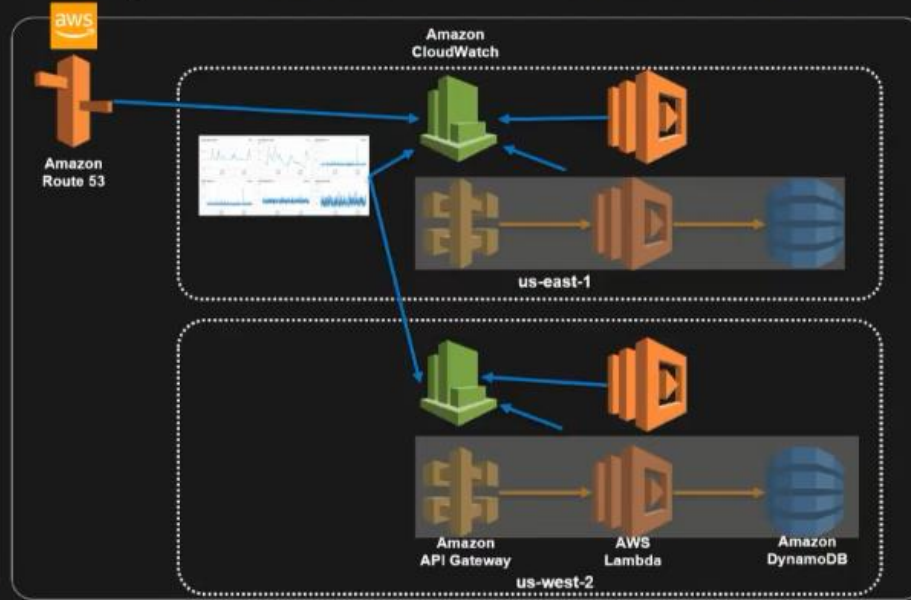
- Pessimistic locking when writing to all stores.
 - Strong consistency
 - Lock contention and potential lower through-put
 - Lower availability
- Quorum based algorithms
 - Eventual consistency but with reduced risk of write conflicts
 - At least 3 replicas
 - Lower availability with certain network partitions
- Conflict-free replicated data type (CRDT)
 - Works well for certain data structure but not others
 - Need to define merge logic for each CRDT

Submit order – architecture highlights

- Active-Active multi-regional API
 - Health checks need to include external dependencies
 - Are your dependencies also multi-region?
- Deploy same stack to multiple AWS Regions
- Data replication using Amazon DynamoDB Streams:
 - Eventual consistency; no two-phase commits.
 - Primary key will include home AWS Region.
 - Leverage semantics of application to simplify replication

Monitoring and logging

Monitoring – Amazon Cloudwatch



AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

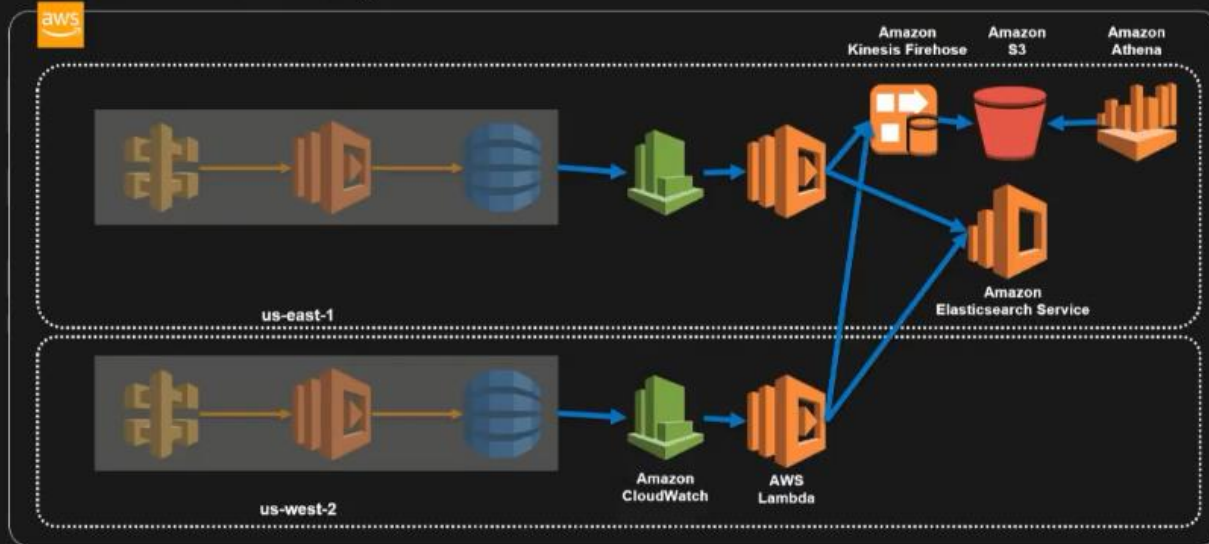


We will use custom metrics and scheduled lambdas to start collecting specific logs off the regions and storing them in S3. We are also collecting the metrics from other regions collected into the us-east-1 region so that we can search in a single place.

Monitoring – CloudWatch highlights

- Granular metrics from within the AWS Region from services
- CloudWatch events with scheduled Lambda to push custom metrics, potentially collecting metrics across AWS Regions
- Amazon Route 53 provides metrics on global health checks
- Dashboard provides single pane of glass across AWS Regions

Centralize logs



aws re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



In each region, we are streaming all the CloudWatch logs coming off our services into a lambda function that are forwarded to a Kinesis Firehose Stream that batches and puts them in a specified S3 bucket and an Elasticsearch service. We can then use Athena to do interactive query on the S3 data.

Deployment

AWS Serverless Application Model (SAM)



AWS CloudFormation extension optimized for serverless

New serverless resource types: functions, APIs, and tables

Supports anything AWS CloudFormation supports

Open specification (Apache 2.0)

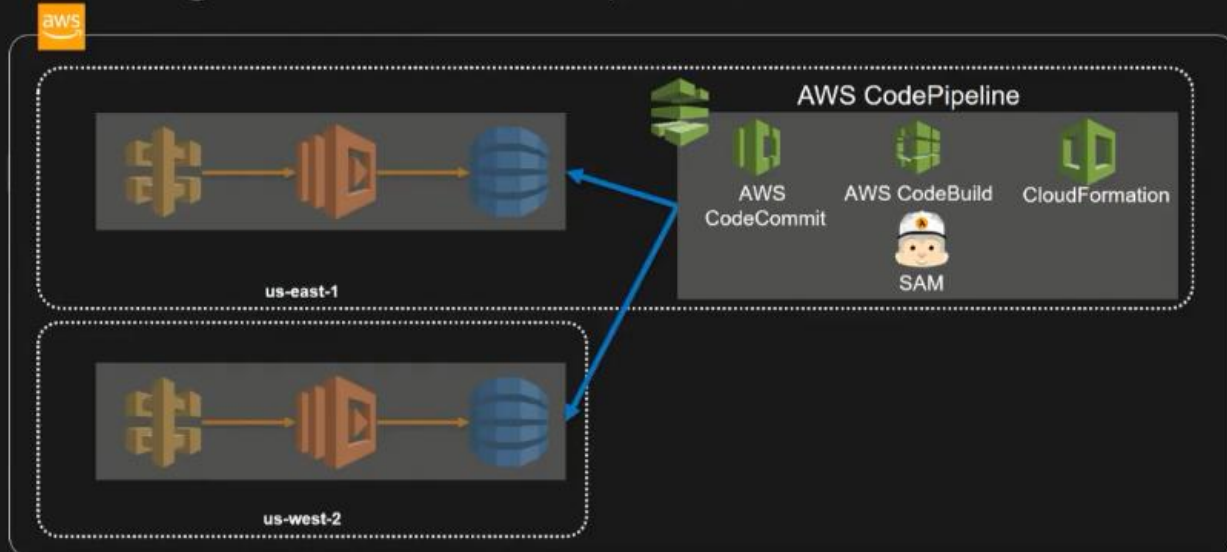
<https://github.com/aws-labs/serverless-application-model>

aws re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Leverage AWS developer tools



AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

Enabling regional endpoints



```
Resources:
  BrowseCatalogApi:
    Type: AWS::Serverless::Api
    Properties:
      DefinitionUri: browse-catalog-swagger.yaml
      StageName: prod
      EndpointConfiguration:
        Types:
          - REGIONAL
      Variables:
        GetListLambda: !Ref GetList
        GetRestaurantLambda: !Ref GetRestaurant
        GetMenuLambda: !Ref GetMenu

  BrowseCatalogDomainName:
    Type: 'AWS::ApiGateway::DomainName'
    Properties:
      CertificateArn: !Ref myCertificate
      DomainName: !Ref domainName
      EndpointConfiguration:
        Types:
          - REGIONAL
```



Region 1
Endpoint

Region 2
Endpoint

```
Resources:
  Region1EndpointRecord:
    Type: AWS::Route53::RecordSet
    Properties:
      Region: "us-east-1"
      HealthCheckId: !Ref HealthcheckRegion1
      SetIdentifier: "endpoint-region1"
      HostedZoneName: !Ref HostedZoneName
      Name: !Ref MultiregionEndpoint
      Type: CNAME
      ResourceRecords:
        - !Sub "${https://${Region1Endpoint}}/prod/"

  Region2EndpointRecord:
    Type: AWS::Route53::RecordSet
    Properties:
      Region: "us-west-2"
      HealthCheckId: !Ref HealthcheckRegion2
      SetIdentifier: "endpoint-region2"
      HostedZoneName: !Ref HostedZoneName
      Name: !Ref MultiregionEndpoint
      Type: CNAME
      ResourceRecords:
        - !Sub "${https://${Region2Endpoint}}/prod/"
```

AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

These are examples of what the SAM templates look like for the BrowseCatalogApi definition.

Summary

SAM is how we recommend deploying your serverless applications, it is a good way to make your stacks look the same across all regions.

Takeaways

- Multi-region architectures add **significant complexity** to your stack – think carefully about your services and whether they need to be multi region at all
- Master/master replication with eventual consistency may create a **worse customer experience** than losing the entire order – full regional outages are highly unlikely, keep it simple!
- Route 53 can perform failover checks and **most operation can be automated** – this does not exonerate operators and developers from the job of monitoring their service. Create relevant metrics and set alarm on them.

Next steps

- Learn more from our “Building a Multi-region Serverless Application with Amazon API Gateway and AWS Lambda” blog post: <https://aws.amazon.com/blogs/compute/building-a-multi-region-serverless-application-with-amazon-api-gateway-and-aws-lambda/>
- API Gateway regional endpoints give you control over your DNS records:
<https://docs.aws.amazon.com/apigateway/latest/developerguide/create-regional-api.html>

Serverless Track

AWS
re:Invent

Thank you!

Magnus Bjorkman, AWS Solution Architect

Stefano Buliani, AWS Specialist Solution Architect

AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

