AWS re:Invent

SVS213-R

Thinking serverless: From business problem to serverless solution

James Beswick

Senior Developer Advocate, AWS Serverless

Amazon Web Services

AWS re:Invent

aws

About me

- James Beswick
  - Email: jbeswick@amazon.com
  - Twitter: @jbesw
- Senior Developer Advocate – AWS Serverless
- Serverless geek
- Software developer and product manager
- Previously:
  - Multiple-startup tech guy
  - Rackspace, USAA, Morgan Stanley, J.P. Morgan …
  - AWS customer since 2012

Agenda

- Where we have come from – Servers
- Where to start with serverless
- Good practices of serverless design
- Whiteboarding

Servers

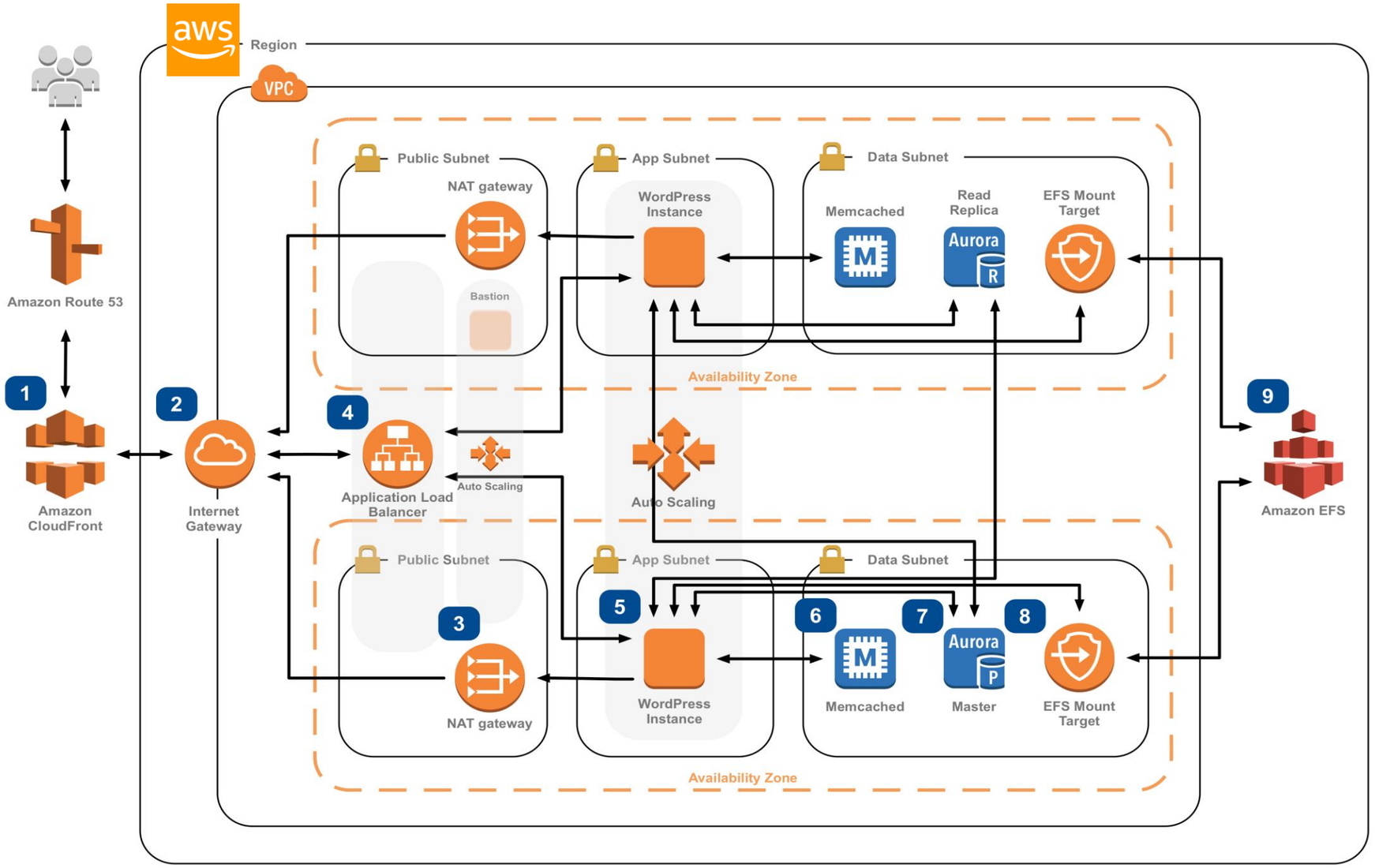
How do we use servers?

- State management
- Monolithic container for functionality
- One version, one server
- Server is an atomic unit of thinking
- The challenges of this model

WordPress Hosting

How to run WordPress on AWS

WordPress is one of the world's most popular web publishing platforms, being used to publish 27% of all websites, from personal blogs to some of the biggest news sites. This reference architecture simplifies the complexity of deploying a scalable and highly available WordPress site on AWS.



1. Static and dynamic content is delivered by **Amazon CloudFront**.
2. An **Internet gateway** allows communication between instances in your VPC and the Internet.
3. **NAT gateways** in each public subnet enable Amazon EC2 instances in private subnets (App & Data) to access the Internet.
4. Use an **Application Load Balancer** to distribute web traffic across an Auto Scaling Group of Amazon EC2 instances in multiple AZs.
5. Run your WordPress site using an **Auto Scaling group** of **Amazon EC2 instances**. Install the latest versions of WordPress, Apache web server, PHP 7, and OPcache and build an Amazon Machine Image that will be used by the Auto Scaling group launch configuration to launch new instances in the Auto Scaling group.
6. If database access patterns are read-heavy, consider using a WordPress plugin that takes advantage of a caching layer like **Amazon ElastiCache** (**Memcached**) in front of the database layer to cache frequently accessed data.
7. Simplify your database administration by running your database layer in **Amazon RDS** using either Aurora or MySQL.
8. Amazon EC2 instances access shared WordPress data in an Amazon EFS file system using **Mount Targets** in each AZ in your VPC.
9. Use **Amazon EFS**, a simple, highly available, and scalable network file system so WordPress instances have access to your shared, unstructured WordPress data, like php files, config, themes, plugins, etc.

aws

AWS Reference Architectures

Where to start with serverless

Understanding how AWS Lambda fits in

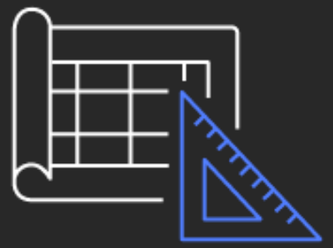
Attributes:

- Runs on demand
- Supports many runtimes
- Responds to events
- Stateless
- Automatically scales

Best practices:

- Avoid lifting-and-shifting
- One Lambda function per purpose
- Keep functions small
- Choose the right runtime for the job
- Use your functions for business logic and plumbing between services

General approach to thinking serverlessly



Features first

Avoid monolithic thinking

Focus on events

Events are triggers that cause action

Statelessness

The key to scaling effectively

Data flow

Make data decisions early on

Use the services

Don’t reinvent the wheel

What are serverless services?

Amazon EventBridge

Amazon Simple Notification Service (Amazon SNS)

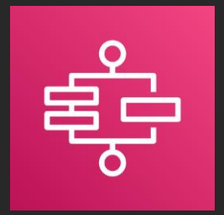
Amazon Simple Queue Service (Amazon SQS)

Amazon API Gateway

Amazon DynamoDB

AWS Lambda

Amazon S3



AWS Step Functions

Amazon Kinesis

AWS IoT Core

Amazon Elastic Transcoder

Integrating with other AWS services

Amazon Rekognition

Amazon Comprehend

Amazon Textract

Amazon Transcribe

Amazon Translate

Good serverless practices

- Infrastructure is disposable
- Asynchronous versus synchronous processing
- You can mix and match runtimes
- Security still matters—build in from the start
- Automation …

Introducing AWS SAM

AWS Serverless Application Model (SAM)

- AWS CloudFormation extension optimized for serverless
- New serverless resource types: functions, APIs, tables
- Supports anything AWS CloudFormation supports
- Open specification (Apache 2.0)

https://github.com/awslabs/serverless-application-model

AWS SAM template

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31

Parameters:
  InputBucketName:
    Type: String
    Default: 's3-auto-translator'
Resources:
  InputS3Bucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: !Ref InputBucketName
  TranslatorFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: translatorFunction/
      Handler: app.handler
      Runtime: nodejs10.x
      MemorySize: 128
      Environment:
        Variables:
          targetLanguage: "es fr de"
      Policies:
        - S3CrudPolicy:
            BucketName: !Ref InputBucketName
      Events:
        FileUpload:
          Type: S3
          Properties:
            Bucket: !Ref InputS3Bucket
            Events: s3:ObjectCreated:*
            Filter:
              S3Key:
                Rules:
                  - Name: suffix
                    Value: '.txt'
```

Tells AWS CloudFormation this is an AWS SAM template it needs to “transform”

Specifies an input parameter

Creates an S3 bucket

Creates a Lambda function with:
- Referenced managed IAM policy
- Language runtime/memory
- Code at the referenced zip location

Defines the event triggering the Lambda function:
- New objects
- Specifies rule (ends in .txt)

AWS SAM template

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31

Parameters:
  InputBucketName:
    Type: String
    Default: 's3-auto-translator'
Resources:
  InputS3Bucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: !Ref InputBucketName
  TranslatorFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: translatorFunction/
      Handler: app.handler
      Runtime: nodejs10.x
      MemorySize: 128
      Environment:
        Variables:
          targetLanguage: "es fr de"
      Policies:
        - S3CrudPolicy:
            BucketName: !Ref InputBucketName
      Events:
        FileUpload:
          Type: S3
          Properties:
            Bucket: !Ref InputS3Bucket
            Events: s3:ObjectCreated:*
            Filter:
              S3Key:
                Rules:
                  - Name: suffix
                    Value: '.txt'
```

Tells AWS CloudFormation this is an AWS SAM template it needs to “transform”

Specifies an input parameter

Creates an S3 bucket

Creates a Lambda function with:
- Referenced managed IAM policy
- Language runtime/memory
- Code at the referenced zip location

Defines the event triggering the Lambda function:
- New objects
- Specifies rule (ends in .txt)

AWS SAM template

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31

Parameters:
  InputBucketName:
    Type: String
    Default: 's3-auto-translator'
Resources:
  InputS3Bucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: !Ref InputBucketName
  TranslatorFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: translatorFunction/
      Handler: app.handler
      Runtime: nodejs10.x
      MemorySize: 128
      Environment:
        Variables:
          targetLanguage: "es fr de"
      Policies:
        - S3CrudPolicy:
            BucketName: !Ref InputBucketName
      Events:
        FileUpload:
          Type: S3
          Properties:
            Bucket: !Ref InputS3Bucket
            Events: s3:ObjectCreated:*
            Filter:
              S3Key:
                Rules:
                  - Name: suffix
                    Value: '.txt'
```

Tells AWS CloudFormation this is an AWS SAM template it needs to “transform”

Specifies an input parameter

Creates an S3 bucket

Creates a Lambda function with:
- Referenced managed IAM policy
- Language runtime/memory
- Code at the referenced zip location

Defines the event triggering the Lambda function:
- New objects
- Specifies rule (ends in .txt)

AWS SAM template

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31

Parameters:
  InputBucketName:
    Type: String
    Default: 's3-auto-translator'
Resources:
  InputS3Bucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: !Ref InputBucketName
  TranslatorFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: translatorFunction/
      Handler: app.handler
      Runtime: nodejs10.x
      MemorySize: 128
      Environment:
        Variables:
          targetLanguage: "es fr de"
      Policies:
        - S3CrudPolicy:
            BucketName: !Ref InputBucketName
      Events:
        FileUpload:
          Type: S3
          Properties:
            Bucket: !Ref InputS3Bucket
            Events: s3:ObjectCreated:*
            Filter:
              S3Key:
                Rules:
                  - Name: suffix
                    Value: '.txt'
```

Tells AWS CloudFormation this is an AWS SAM template it needs to “transform”

Specifies an input parameter

Creates an S3 bucket

Creates a Lambda function with:
- Referenced managed IAM policy
- Language runtime/memory
- Code at the referenced zip location

Defines the event triggering the Lambda function:
- New objects
- Specifies rule (ends in .txt)

AWS SAM template

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31

Parameters:
  InputBucketName:
    Type: String
    Default: 's3-auto-translator'
Resources:
  InputS3Bucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: !Ref InputBucketName
  TranslatorFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: translatorFunction/
      Handler: app.handler
      Runtime: nodejs10.x
      MemorySize: 128
      Environment:
        Variables:
          targetLanguage: "es fr de"
      Policies:
        - S3CrudPolicy:
            BucketName: !Ref InputBucketName
      Events:
        FileUpload:
          Type: S3
          Properties:
            Bucket: !Ref InputS3Bucket
            Events: s3:ObjectCreated:*
            Filter:
              S3Key:
                Rules:
                  - Name: suffix
                    Value: '.txt'
```

Tells AWS CloudFormation this is an AWS SAM template it needs to “transform”

Specifies an input parameter

Creates an S3 bucket

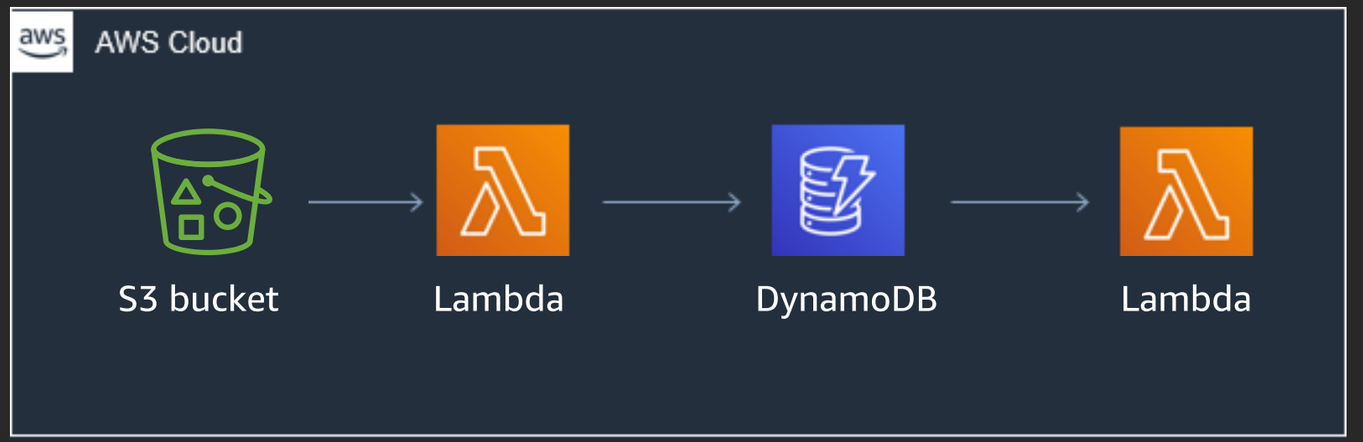
Creates a Lambda function with:
- Referenced managed IAM policy
- Language runtime/memory
- Code at the referenced zip location

Defines the event triggering the Lambda function:
- New objects
- Specifies rule (ends in .txt)

AWS SAM transforms YAML into infrastructure

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31

Parameters:
  InputBucketName:
    Type: String
    Default: 's3-auto-translator'
Resources:
  InputS3Bucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: !Ref InputBucketName
  TranslatorFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: translatorFunction/
      Handler: app.handler
      Runtime: nodejs10.x
      MemorySize: 128
      Environment:
        Variables:
          targetLanguage: "es fr de"
      Policies:
        - S3CrudPolicy:
            BucketName: !Ref InputBucketName
      Events:
        FileUpload:
          Type: S3
          Properties:
            Bucket: !Ref InputS3Bucket
            Events: s3:ObjectCreated:*
            Filter:
              S3Key:
                Rules:
                  - Name: suffix
                    Value: '.txt'
```



Whiteboarding

AWS re:Invent

aws

1. Form upload

Create a serverless application to support a customer review form submitted from a webpage

- Incoming responses must be translated to English
- Allow user to upload image with a response
- Email any negative comments immediately
- Only allow signed-in users to post reviews

2. Create a virtual queue

A global retailer wants to implement a virtual queue for customers. Use an IoT button to manage a “now serving” counter.

- Create a real-time display showing the current count
- Speak the number in the language of the local office
- Send a daily click-history email report
- Publish the current count into a mobile app

3. Web application

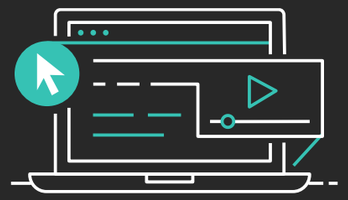
Create a serverless web application to support a national e-commerce business

- Ensure fast performance for visitors in multiple regions
- Allow users to create accounts (including social login)
- Support uploading and serving user videos
- Let users “like” products and receive updates

Wrap-up

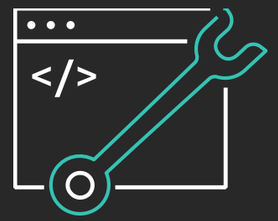
Learn serverless with AWS Training and Certification

Resources created by the experts at AWS to help you learn modern application development



Free, on-demand courses on serverless, including

- Introduction to Serverless Development
- Getting into the Serverless Mindset
- AWS Lambda Foundations
- Amazon API Gateway for Serverless Applications
- Amazon DynamoDB for Serverless Architectures



Additional digital and classroom trainings cover modern application development and computing

Visit the Learning Library at https://aws.training

aws training and certification

Thank you!

James Beswick

jbeswick@amazon.com
jbesw@

AWS re:Invent

aws

Please complete the session survey in the mobile app.