XSpec in the Cloud with Diamonds

Sandro Cirulli, XSpec <sandro@sandrocirulli.net>

Abstract

Running XSpec tests in a development team is usually performed via a CI server/service. However, this comes with limitations related to the use of private repositories and to the cost and burden of administering CI servers.

This paper describes an alternative approach for running XSpec tests from private repositories using a serverless architecture built on AWS Lambda. It describes the technical configuration and discusses the benefits, cost optimization, and constraints of a serverless architecture for running XSpec tests.

Table of Contents

Introduction

XSpec is a unit test and behaviour-driven development framework for XSLT, XQuery, and Schematron [xspec]. XSpec test suites are generally executed:

- On a local machine using the shell or batch scripts, the oXygen XML editor, etc.
- On a Continuous Integration (CI) server using software tools like Jenkins or online CI services like Travis, CircleCI, AppVeyor, etc.

While running tests locally is typically performed by individual developers during their development process, running tests on a CI server is usually employed within a team to integrate code changes under version control.

CI servers like Jenkins offer fine grained control on how the test suite is executed and support both private and public version controlled repositories. However, this configuration requires a server to run the CI software and a system administrator to maintain both the server and the software.

Conversely, online CI services like Travis run entirely on the cloud and do not require system administration or server maintenance. On the other hand, these CI services are free to use only for open source projects in public repositories and charge fees for private repositories and improved capabilities.

The aim of this paper is to show an alternative approach using a serverless architecture built on AWS Lambda. This allows to run XSpec tests from private repositories while keeping costs low and avoiding server and software maintenance.

AWS Lambda and Serverless Architecture

Lambda is a compute service provided by Amazon Web Services (AWS) allowing to run code without provisioning or managing servers [lambda]. AWS Lambda claims to scale applications and workloads automatically from few requests per day to thousands per second and charges only for the compute time used.

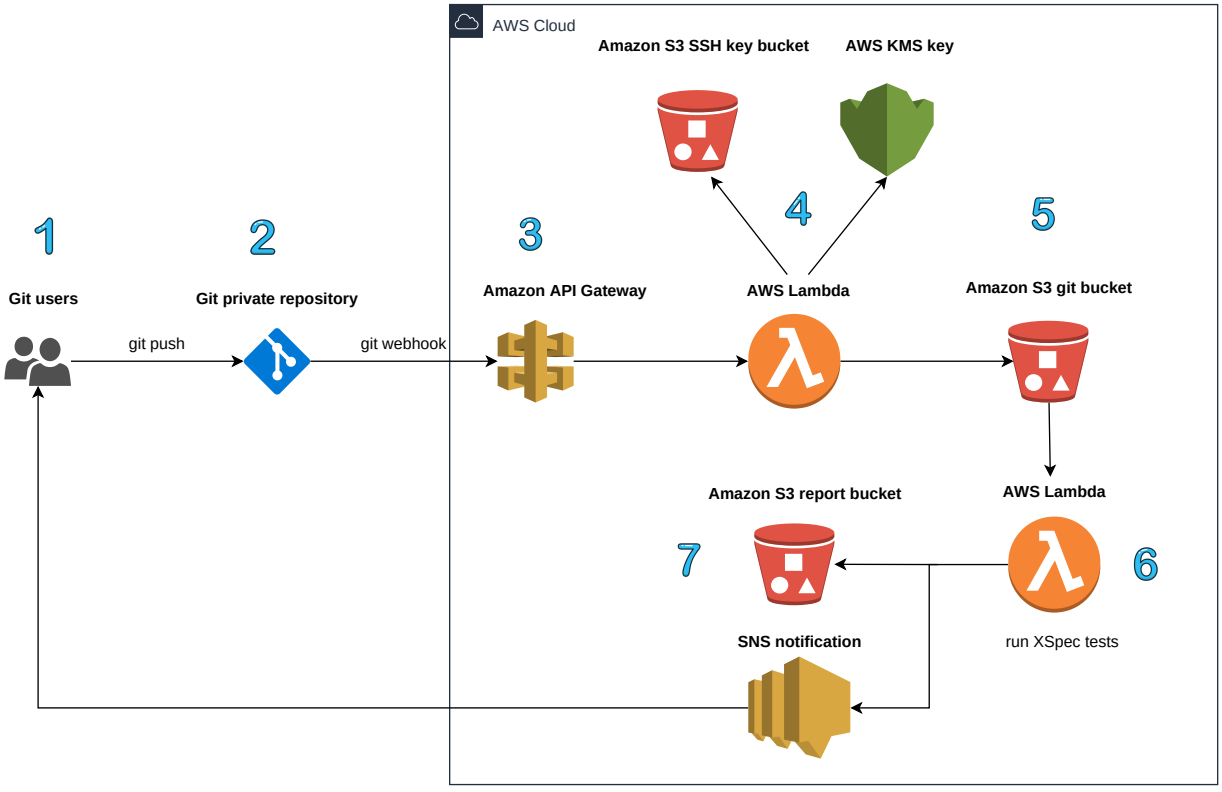
AWS Lambda can be used in conjunction with other services to build a serverless architecture. This is a cloud architecture typically running a function inside a stateless computing environment triggered by an event [fowler]. Serverless architectures enable to run applications and workloads without managing server infrastructure and with reduced operational costs and flexible scaling [cncf].

In the next section I am going to explain how to build a serverless architecture for running XSpec test suites triggered by an event such as pushing new code to a version control system.

Technical Configuration

Figure 1, “Workflow for running XSpec tests in a serverless architecture” illustrates the workflow for running XSpec tests in a serverless architecture.

Figure 1. Workflow for running XSpec tests in a serverless architecture



The workflow comprises the following steps:

1. A developer pushes a code change into a private git repository.

2. A git webhook sends a payload to an Amazon API gateway.

3. The API gateway endpoint accepts the webhook request from git and triggers a Lambda function.
4. The Lambda function connects over SSH to the git service. SSH private keys are stored securely using Amazon S3 and AWS KMS.
5. The zipped content of the git repository is stored in S3.
6. The S3 storage event triggers a lambda function that unzips the content of the git repository and executes XSpec tests.
7. The report of the XSpec tests is stored in S3. A notification is sent via email through SNS to the development team.

For the readers unfamiliar with AWS, I shortly describe the purpose of each Amazon service mentioned in the diagram and in the next sections:

- **Amazon API Gateway**: a managed service allowing to create API endpoints [api_gateway].
- **Amazon S3**: an object storage service (S3 stands for Simple Storage Service) [s3].
- **Amazon KMS**: a managed service for creating, storing and accessing encryption keys (KMS stands for Key Management Service) [kms].
- **AWS Lambda**: a service for building serverless applications and running code without managing servers [lambda].
- **Amazon SNS**: a messaging service for sending notifications such as emails, sms, etc. to subscribing clients (SNS stands for Simple Notification Service) [sns].
- **AWS CloudFormation**: a service for describing and provisioning infrastructure resources in AWS [cloudformation].
- **AWS IAM**: a service for controlling access to AWS resources (IAM stands for Identity and Access Management) [iam].
- **Amazon CloudWatch**: a service for monitoring AWS resources [cloudwatch].
- **AWS Billing and Cost Management**: a service for paying, monitoring, and budgeting costs in AWS [billing].

In the next sections I am going to describe how to set up and configure the serveless architecture. First, I am going to link the private git repository with S3 in order to store changes whenever a user pushes new commits to git (steps 1 to 5). Then I'm goint to configure the lambda function to run XSpec tests and send notifications to the user (steps 6 and 7). Whenever possible, I will refer to code in my GitHub account so that readers wishing to replicate this can get hold of code examples and templates.

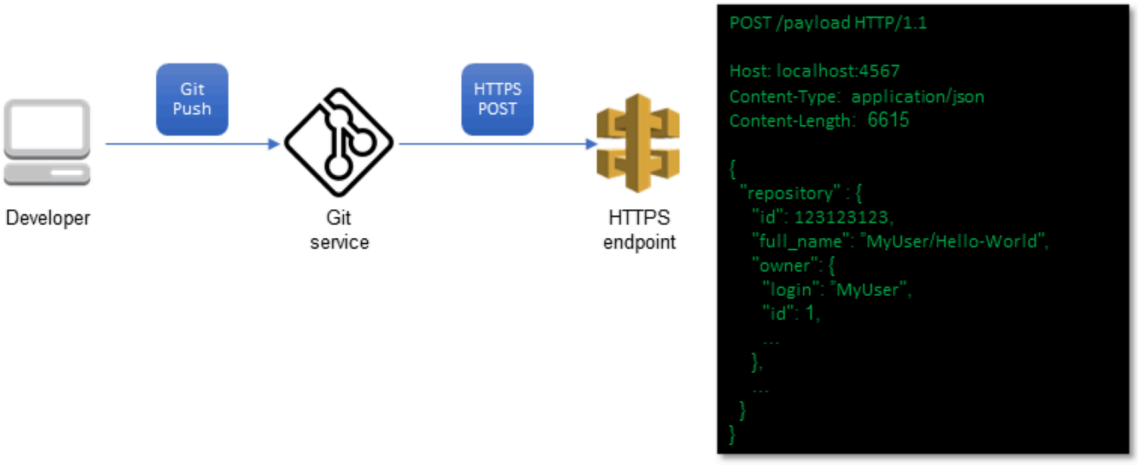
Linking Git to S3

The easiest way to link git repositories to S3 and build the first part of the infrastructure in the diagram is to use the Quick Start deployment guide provided by AWS [git2s3]. This contains a quick start button allowing to start all the necessary AWS resources with a single click using an AWS CloudFormation template. The process is thoroughly documented in the guide which also describes several options for customizing the configuration.

In my GitHub repository [markupuk2019] I wrote additional documentation for my configuration. In particular, I set up a GitHub private repository and used a GitHub webhook to send a payload to the AWS

API Gateway whenever a new commit is pushed to the GitHub repository (this option is referred to as git pull endpoint in the Quick Start guide). Figure 2, “Workflow for triggering the webhook”, taken from the official Quick Start documentation, illustrates the webhook workflow.

Figure 2. Workflow for triggering the webhook



This configuration covers points 1 to 5 of the diagram. At the end of the process the GitHub private repository is replicated in a zip file within S3.

Lambda Configuration

This configuration covers point 6 and 7 of the diagram. In particular, it describes the configuration for setting up the lambda function running XSpec tests. This entails:

1. Retrieving the GitHub code stored in a zip file in S3.
2. Unzipping the file and storing it in a local directory accessible by the lambda function.
3. Setting up environment variables required by XSpec (i.e. paths to Saxon and HTML report).
4. Running the XSpec test suite for all the XSpec tests stored in a given directory.
5. Storing the HTML report in S3.
6. Notifying the developer in case a test failed and providing a link for accessing the HTML report in S3.

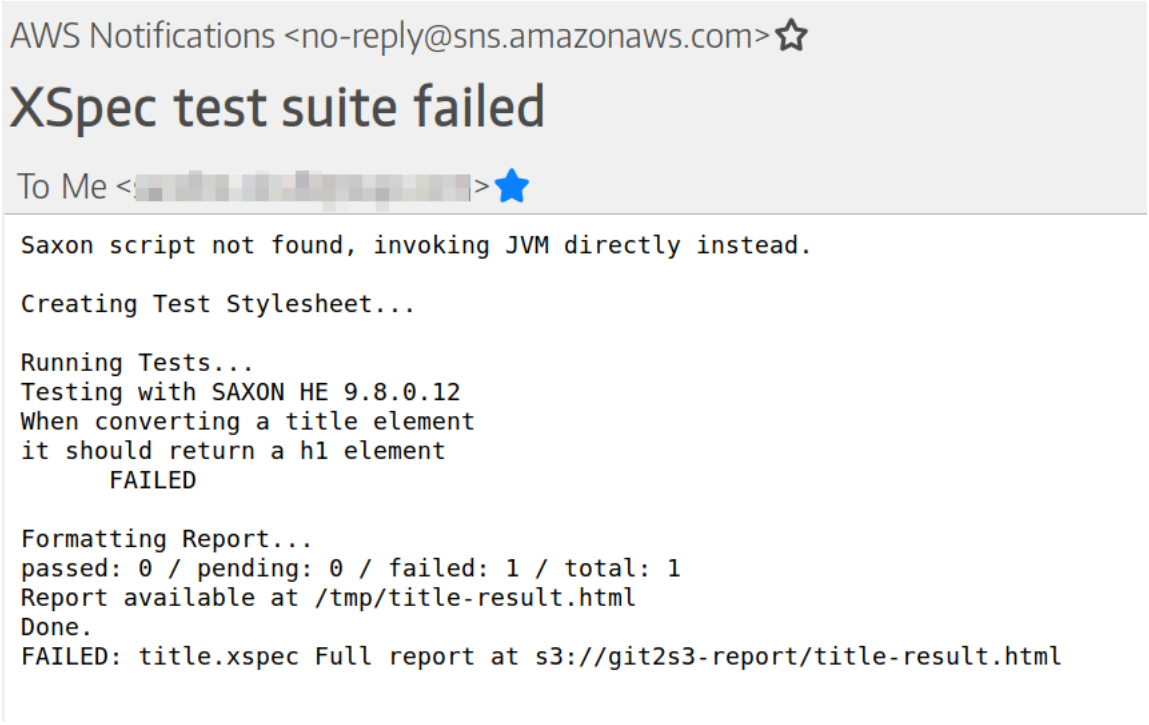
A step-by-step guide to replicate this configuration and the code for the lambda function is available in my GitHub account [markupuk2019]. It is worth highlighting the following points:

- **Custom Runtime**: AWS Lambda natively supports lambda functions written in Java, Go, PowerShell, Node.js, C#, Python, and Ruby. AWS Lambda also provides a Runtime API allowing to implement the lambda function in any programming language [custom_runtime]. I used the latter approach and implemented the lambda function in bash as calling the `xspec.sh` shell script is the simplest way to run the XSpec test suite. However, the lambda function could be re-written using virtually any programming language.
- **Layers**: the lambda function makes use of layers [layers]. A layer is a zip file containing additional libraries and dependencies. In particular, I used layers for the XSpec runtime, the Saxon HE jar file, and

bash [bash_layer]. It is also possible to add a layer for Apache Ant and run the XSpec test suite via Ant which can speed up the test suite execution.

- **Memory and Time Settings**: these are configured in the settings for the lambda function. As running large XSpec test suites can be memory and time intensive, I recommend to adapt the values of memory and timeout accordingly. In my experience 512 MB memory and 2 minute timeout is a minimum threshold for running few XSpec tests.
- **IAM configuration**: the lambda function is granted permissions to interact with other AWS services like S3 and SNS via a IAM role [iam_role]. It is a good security practice to allow access only to the relevant S3 buckets.
- **Notification**: I configured the lambda function to send email notifications via SNS (Figure 3, “Email notification” shows an example of email notification with a failed test). It is also possible to configure SNS to send notifications to a chat messaging service like Slack and this may be more appropriate for large development teams.

Figure 3. Email notification



- **CI Workflow**: the lambda function is configured to stop as soon as a test fails: this is done in order to reduce the execution time and to lower costs. However, it is possible to adapt the lambda function to run the full test suite and report all the failing tests in the notification.
- **Troubleshooting**: debugging a lambda function can be challenging as the environment upon which it runs is stateless. Using CloudWatch to monitor the execution of the function and outputting the results of commands to the CloudWatch logs is extremely useful for troubleshooting.

Analysis and Discussion

In this section I analyze and discuss the major benefits, constraints and limitations of the serverless architecture for running the XSpec test suites.

Benefits of Serverless Architecture

Not having to provision and maintain servers and the software running on it is by far the greatest advantage of running a serverless architecture. Not only this can help reducing operational costs but it also frees up software engineering time that could be spent in more valuable tasks.

Another advantage is the scalability of a serverless environment once it is well architected in independent components. In fact, a serverless architecture is stateless and event-driven and can be easily scaled up and down by adjusting the resources and parameters of the single components. For example, memory allocation is a parameter in the lambda function; running out of storage space is not an issue since S3 provides virtually infinite storage space.

Finally, the serverless architecture is highly available since it relies on AWS managed services like API Gateway, Lambda, and S3: the architecture will go down only if and when AWS experiences an outage.

Memory and Time Execution Constraints

As described in the lambda configuration section, running XSpec test suites can be a memory and time intensive process and parameters for memory allocation and timeout need to be adjusted according to the number and type of XSpec tests.

AWS Lambda enables to run a function for up to 15 minutes, after which the function will automatically timeout. Therefore the XSpec test suite needs to run within this time constraint. This limitation is useful as it helps keeping the test suite manageable and its execution fast enough to provide feedback to developers within a short time frame. As the test suite becomes more complex and more tests are added, it is recommended to break it down into different groups and assign a lambda function for each group of XSpec tests. This offers the advantage of running multiple lambda functions in parallel and keeps the total execution within the 15 minute time constraint.

Cost Optimization

Reducing operational costs is one of the major advantages of a serverless solution. However, costs in a cloud computing environment need to be closely monitored in order to avoid surprises.

AWS provides a billing console for monitoring and estimating costs. It also provides billing alarms when AWS costs go over a certain threshold. These are very useful tools for keeping the cost within the allocated budget.

Costs for AWS Lambda are based on usage and charged according to the number of requests and the duration [lambda_pricing]. At the time of writing AWS provides a generous free tier of 1 million free requests and 400,000 GB-seconds of compute time per month. This makes a test suite based on a serverless architecture particularly attractive as it can be used with very little operational costs. However, increasing the memory and the timeout settings of the lambda function also increases the costs so these need to be factored in.

Vendor Lock-in

Amazon introduced AWS Lambda in 2014 and was the first cloud provider to offer and popularize this type of service in its cloud computing platform. However, nowadays other major cloud providers offer similar options for building serverless architecture: for example Microsoft with Azure Functions and Google with Google Cloud Functions. However, building serverless applications with a cloud provider inevitably comes with a degree of vendor lock-in as serverless services like AWS Lambda are proprietary and cannot be easily ported to another cloud provider. Therefore building serverless architecture may increase the depedency towards a single cloud provider.

Conclusion

In this paper I described an alternative approach for running XSpec test suites using a serverless architecture built on AWS. The high level configuration is explained in this paper and more details with step-by-step instructions and code examples are available on my GitHub account. I also analyzed and discussed benefits and constraints of a serverless solution.

I hope this can be helpful for development teams wishing to implement a CI workflow while reducing operational burden and costs. I would be interested in knowing and possibly helping anyone willing to implement this serverless workflow for running XSpec tests.

Bibliography

[api_gateway] Amazon Web Services, Inc.. *Amazon API Gateway*. https://aws.amazon.com/api-gateway . Accessed: 26 May 2019.

[bash_layer] GitHub. *Bash Lambda Layer*. https://github.com/gkrizek/bash-lambda-layer . Accessed: 26 May 2019.

[billing] Amazon Web Services, Inc.. *What Is AWS Billing and Cost Management?*. https://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/billing-what-is.html . Accessed: 26 May 2019.

[cloudformation] Amazon Web Services, Inc.. *AWS CloudFormation*. https://aws.amazon.com/cloudformation . Accessed: 26 May 2019.

[cloudwatch] Amazon Web Services, Inc.. *Amazon CloudWatch*. https://aws.amazon.com/cloudwatch . Accessed: 26 May 2019.

[cncf] Cloud Native Computing Foundation. *CNCF WG-Serverless Whitepaper v1.0*. https://github.com/cncf/wg-serverless/raw/master/whitepapers/serverless-overview/cncf_serverless_whitepaper_v1.0.pdf . Accessed: 26 May 2019.

[custom_runtime] Amazon Web Services, Inc.. *Custom AWS Lambda Runtimes*. https://docs.aws.amazon.com/lambda/latest/dg/runtimes-custom.html . Accessed: 26 May 2019.

[fowler] Martin Fowler. *Serverless Architectures*. https://martinfowler.com/articles/serverless.html . Accessed: 26 May 2019.

[git2s3] Amazon Web Services, Inc.. *Git Webhooks with AWS Services*. https://aws-quickstart.s3.amazonaws.com/quickstart-git2s3/doc/git-to-amazon-s3-using-webhooks.pdf . Accessed: 26 May 2019.

[iam] Amazon Web Services, Inc.. *AWS Identity and Access Management*. https://aws.amazon.com/iam . Accessed: 26 May 2019.

[iam_role] Amazon Web Services, Inc.. *IAM Roles*. https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html . Accessed: 26 May 2019.

[kms] Amazon Web Services, Inc.. *AWS Key Management Service (KMS)*. https://aws.amazon.com/kms . Accessed: 26 May 2019.

[lambda] Amazon Web Services, Inc.. *What is AWS Lambda?*. https://docs.aws.amazon.com/lambda/latest/dg/welcome.html . Accessed: 26 May 2019.

[lambda_pricing] Amazon Web Services, Inc.. *AWS Lambda pricing*. https://aws.amazon.com/lambda/pricing . Accessed: 26 May 2019.

[layers] Amazon Web Services, Inc.. *AWS Lambda Layers*. https://docs.aws.amazon.com/lambda/latest/dg/configuration-layers.html . Accessed: 26 May 2019.

[markupuk2019] GitHub. *Markup UK 2019*. https://github.com/cirulls/markupuk2019 . Accessed: 26 May 2019.

[s3] Amazon Web Services, Inc.. *Amazon S3*. https://aws.amazon.com/s3 . Accessed: 26 May 2019.

[sns] Amazon Web Services, Inc.. *Amazon Simple Notification Service*. https://aws.amazon.com/sns . Accessed: 26 May 2019.

[xspec] GitHub. *XSpec*. https://github.com/xspec/xspec . Accessed: 26 May 2019.