Serverless, Schmerverless!

One developer’s search for knowledge in the hazy world of Serverless software development

About Me

- Ken Rimple
  - Director of Training and Mentoring, Chariot Solutions
  - Present about emerging / emergent tech topics
  - Mentor consultants, clients in a wide array of technologies
  - Focus on Cloud, container-based technologies and Single Page Applications in React, Angular, Vue

What we won’t discuss

- Lambda Layers
- Production tuning
- Complex configuration
- API Gateway 2.0
- CDK or Terraform
- Websockets or advanced configurations

Serverless Software Development is…

- Difficult to master
- Not well-documented
- A super leaky abstraction
- Fraught with complex choices (CORS anyone?)

So why do it?

Figure 1. available in the Slack channels as :slam: courtesy Martin Snyder

- Move quickly and get stuff done?
- Avoid paying for idle servers
- Avoid managing server OSes, application servers
- Lower costs than legacy application?

Some Serverless options

- Manually, via CloudFormation or Terraform
- Using AWS Serverless SAM
- Using the Serverless Framework
- Using a stronger abstraction, such as Architect

Serverless, or Schmerverless?

EC2 Apps	Container Apps	Serverless Apps
Virtualized Servers	Distributed, multi-node platform	Functions in the Cloud
Easy to host existing servers	Run the same platform on desktops, production	Runtime managed by AWS, harder to test locally
Can be costly to host (overprovisioning, paying for idle time)	Can scale quickly up/down based on demand (Fargate)	Only pay based on usage
Can be cloud agnostic	Can be cloud agnostic	Are tightly locked to AWS

Lambda Functions in AWS

AWS Lambdas

Lambdas functions have a common signature:

```
function handler(event, context) {
  // process input in event, refer to context
  // then...
  return {
    statusCode: 200,
    body: 'OK'
  };
}
```

- Lambdas are deployed on AWS and have an ARN
- They access “Serverless” services or even exposed servers like RDS
- AWS Lambda runtimes span all major programming languages

Serverless doesn’t mean no servers. It just means 'not your servers!'

Lambda’s division of labor

You	AWS
Deploy Lambda code	Launch and prune Lambda runtimes
Define IAM Roles, Permissions	Protect based on Roles Permissions
Map Lambdas to events including API Gateway URLs, AWS infrastructure	Execute the Lambdas when events are triggered
Define OAuth, Cognito auth	Protect via the configured authorization
Allocate memory to the Lambda, set a timeout, potentially keep warmed	Allocates up to 1 CPU per executing Lambda

Lambda services are tuned by tweaking memory allocation, concurrent instances, and setting warm-up settings

AWS Lambda Functions can…

- Respond to HTTP endpoints via API Gateway or Application Load Balancer
- Act as microservices to replace monolithic APIs
- Respond to events in AWS
  - Addition/removal of a file to S3
  - Transform incoming data in Kinesis, other systems
  - Execute on a trigger in an Amazon database row
  - Respond to a GraphQL query

Lambdas exist to execute event-driven logic and communicate with Serverless APIs

Serverless services…

A Serverless Service is...

- Billed for only when consumed
- Not truly "serverless"
- Minimally configurable
- Usually accessed from the Lambda via the AWS SDK

Manually building Lambda-based Microservices

- CloudFormation is a very verbose and tedious API
- But you need to learn it if you work with serverless AWS
- It defines resources, wires them together
- Deployments result in Stacks
  - Stacks can be queried, updated, dropped
  - Stacks can accept input parameters, and output values
  - Stacks can contain just about any AWS service

CF is the basis of SAM and is used by Serverless

A CloudFormation Snippet

Sample listing

```
  APIGateway: ①
    Type:                        "AWS::ApiGateway::RestApi" ②
    Properties:
      Name:                      !Sub "${AWS::StackName}"   ③
      Description:               "Endpoint"
      EndpointConfiguration:
        Types:                   [ "REGIONAL" ]
```

1. Each configured resource has an identifer (ARN)
2. Types are well-defined and used to build infrastructure
3. CloudFormation has functions and substitution variables

This goes on for 579 lines in Two S3 Buckets, A Lambda, and infrastructure

To build a Lambda HTTP service, create

Pure CloudFormation Pros/Cons

Pro	Con
Mature, used by thousands of applications	Verbose, confusing, takes a long time to learn and master
Complete control over options you can set	No significant logic beyond some conditional configuration
Inputs/Outputs and Exports provide linkages between CF Templates	Manual changes create drift, which can cause stack updates to fail
Can deploy the same stack over and over with different parameters	Tooling is not great, and can lag behind types available

Serverless development platforms

My criteria for a framework or tooling around Serverless

- It should provide an improvement to provisioning over pure CloudFormation
- It should be well documented
- It should have wide adoption
- It should have great examples
- It should not increase cognitive load

⚠ *No one tool has all of these criteria covered!*

Ken’s "things to learn" for success in Serverless regardless of tool

- AWS CLI configuration including credential management
- Understanding AWS Accounts and the AWS Console
- Basic understanding of IAM Users, Roles, Policies, Permissions
- Basic CloudFormation skills - including how to create objects, and use functions like !GetAtt, !Ref, !Sub, etc.
- Understanding about linking CF stacks together with Inputs, Outputs, Exports
- Basic Lambda execution in your language, as well as managing dependencies, how it logs output
- Enough API Gateway to hurt yourself including turning on CloudWatch logging and X-Ray
- Enough time for the project team to learn all of this before assuming heavy project deadlines

Serverless development platforms for this talk

- AWS Serverless Application Model (SAM)
- The Serverless Framework
- Architect

There are other options out there, we've only got time for 3

AWS Serverless Application Model

AWS Serverless SAM

- The Serverless Application Model framework from AWS, provides:
  - Helpful Types to deal with boilerplate config
  - A CLI tool (`aws-sam-cli`)
  - A deployment command
  - Both local and cloud invocation options

Your cognitive load is a bit reduced

- Lambda deployment is automated (guided deploy sets up S3 bucket, stores info in `serverless.toml`)
- Helper types for Lambdas, API Gateway, events, others
- When defining a Serverless Lambda, SAM:
  - Creates a shared IAM Role for the project
  - Grants execute authority to the Role
  - Exposes the Lambda based on the event you attach (REST API, AWS event)
  - Creates an API Gateway if you mount a REST API

Serverless will save you many, many lines of CloudFormation code, but you need to understand what it's doing to debug it… So head to CloudFormation in the Console

Review SAM App

Ken’s SAM tips

- Separate your slow-moving infrastructure from applications
- Avoid explicit resource naming where possible
- If naming, make sure to involve your current stage name (dev, qa, production)
- Link stacks together via exported resources from other CloudFormation templates
- SAM sometimes does not detect stack changes, you may need to force deployments
- Some of the Serverless transform Types lack significant documentation or examples are hard to find

Recommendation: delete .aws-sam and rebuild during a deploy to the cloud to protect yourself

Ken’s SAM pros/cons

Pro	Con
`AWS::Serverless::xxx` types can save time and provide defaults for Lambdas, API Gateway, a simple DynamoDB table, nested applications	You are still mostly rolling out CloudFormation templates
No new syntax to learn - it’s just CloudFormation	Local development is challenging, though possible to a degree
Tooling is a bit better than standard `aws cfn` commands, allows viewing of logs, invoking, etc.	Many features are bewildering to learn at first without a lot of research (API Gateways)

The Serverless Framework

The Serverless Framework

- Transforms a cross-cloud `serverless.yml` YAML file’s syntax into native Cloud platforms
- Provides a cross-cloud app monitoring platform for free, commercial options for enterprise / team integrations (https://serverless.com/pricing/)
- On AWS:
  - CloudFormation is used for the transforms for that platform
  - You can import and use straight CloudFormation YAML or JSON
  - The Serverless YAML file has its own DSL
  - It’s not quite a leakless abstraction as each cloud has different mappings

Serverless does not use the SAM transform classes

Serverless Framework Example Review

Plugins make things interesting

- Modify the build process and extend the platform
  - `serverless-plugin-offline` - emulates lambda and API Gateway locally
  - `serverless-plugin-tracing` - enables deep X-Ray tracing with minimal effort
  - `serverless-finch` - takes a SPA’s distribution directory and pushes it to an S3 bucket, hosts it!
  - `serverless-python-requirements` - uses the `requirements.txt` file to bundle Python dependencies

See https://serverless.com/plugins/ for more details.

Ken’s Serverless tips/comments

- The syntax is just, well, different
- Almost better to learn SAM first, do a project in it, THEN use Serverless
- CORS is bewildering here just like in SAM, but a bit worse
- There are magic components (a pre-mounted Api Gateway) you may need to override
- Serverless CLI is `serverless`, shortened to `sls`
- Serverless may not detect changes in your stack. Drop it and re-add it, or use `--force`
- can’t remove a stack if you have syntax errors in the serverless.yml file
- See the generated CloudFormation with `sls print` - this can help you a lot

Serverless Pros/Cons

Pro	Con
Cross-cloud concepts means learning one provisioning tool	The DSL, Serverless’s own magic variables add cognitive load to CloudFormation
You can still use CloudFormation	Tooling hard to find
Serverless plugins are a great feature	Be aware of plugin rot and pick the highly used ones
The REST-based API Gateway is verbose and confusing	APIGateway V2 support is not baked yet but will cut down complexity

Chariot’s Drew DeCarme used Serverless on his application deploying to CloudFlare

Architect

So, what’s wrong here so far?

- A huge amount of cognitive load to just get started
- The platform itself is very low-level
- Leakier than Niagara Falls for the developer

I called this "Serverless, Schmerverless" because of how much it took to learn enough to be dangerous

We need less complex tooling

- Hide all of the complexity you can
- Give me something I can do without knowing how it is being done
- Make me only care about stuff I need
- Do one thing and do it well

Architect is

- A strong abstraction over Lambdas and Serverless
- Provides easy access to
  - Lambdas
  - Messaging with SNS and SQS
  - Data with DynamoDB tables

Architect is not

- A general-purpose serverless platform
- The first thing you run to to create AWS event integrations
- Easy to integrate with existing data sources in RDS
  - You can use JS-based macros to access AWS::Serverless::Cloudformation instance
  - This should allow you to configure resources otherwise not easily available

Architect Example

You like this?

- Tomorrow at 1:30PM we booked Brian LeRoux for a full-on talk on Architect
  - Less, but Better, Serverless with OpenJS Architect

The land that time did not permit…

Stuff on the cutting room floor follows. Hope it helps!

General Lambda notes

DRY is hard in Lambda

- Use Lambda Layers for shared functions across Lambdas
- Or, import code into Lambdas, but be aware you will increase its size
- If you add libraries
  - Add the smallest ones and least pieces
  - Use a tree shaker, exclude portions, etc

Lambdas and Latency

- Avoid
  - Too many external resources or libraries
  - Too much memory - filter on AWS, not on the client!
  - Too much time - your function can time out if it runs away and cost you money if repeatedly executed
  - Too much code - if you have complex logic and tiers to your application, consider a Docker container and an actual app stack

Use CloudWatch to review your memory usage

After each execution it prints out the usage details

```
REPORT RequestId: f1a419f8-b21c-4ecd-a503-51397d141ba1
  Duration: 176.12 ms
  Billed Duration: 200 ms
  Memory Size: 1152 MB
  Max Memory Used: 90 MB
  Init Duration: 411.44 ms
```

- Tune your memory or adjust your code based on how long / how much it consumes

Debugging / Tracing

- You will spend a lot of time staring at useless stack traces until you learn
  - API Gateway swallows mis-use of responses and requests and bad mappings
  - Turn on X-Ray tracing in API Gateway stages to help find errors
  - Use logging as much as possible in your code to help you figure out what's going on when deployed
  - Test locally (`sam invoke local`) and use your local client stack to get some useful errors
  - Unit test to get your logic debugged before finding out you wrote code that has logic errors
- You will not completely get away from AWS-native concepts even with Architect

Exception handling

- Your errors may be swallowed by the API Gateway in a REST API
- Turn on logging on API Gateway (View your deployed stage, and add logging)
- You can define custom responses for various errors

Final Thoughts

- SAM is the most long-form and native AWS platform
- Serverless is powerful and great for quick attachment of Lambdas to AWS events
- Arc is a rapid development tool for AWS and might be the best for quick apps without lots of external dependencies

I will hang out in the room for Q&A after the talk

TL;DR

- SAM and Serverless are good ways to develop AWS-based serverless applications
- You are going to have to learn and keep track of a LOT of things
- The cognitive load of managing serverless platforms could outweigh the benefits for the team
- The main benefit is the lack of spend when the app is 'idle'
- The main drawback is that the platforms are very immature
- You will tie yourself to AWS deeply, and have challenges with local development compared to Docker-based container development