trek10

Common Serverless Use Cases & Architectures

Andy Warzon
Founder & CEO, Trek10

Jared Short
Director of DevOps, Trek10

amazon web services | Partner Network
ADVANCED CONSULTING PARTNER

Serverless? But what about...

salesforce

HEROKU

Too Curated
Fat Clients
Wrong unit: Deployed (Monolithic) Applications

Serverless = Services and Functions are the Platform Abstraction and the Unit of Deployment

In AWS, Serverless =

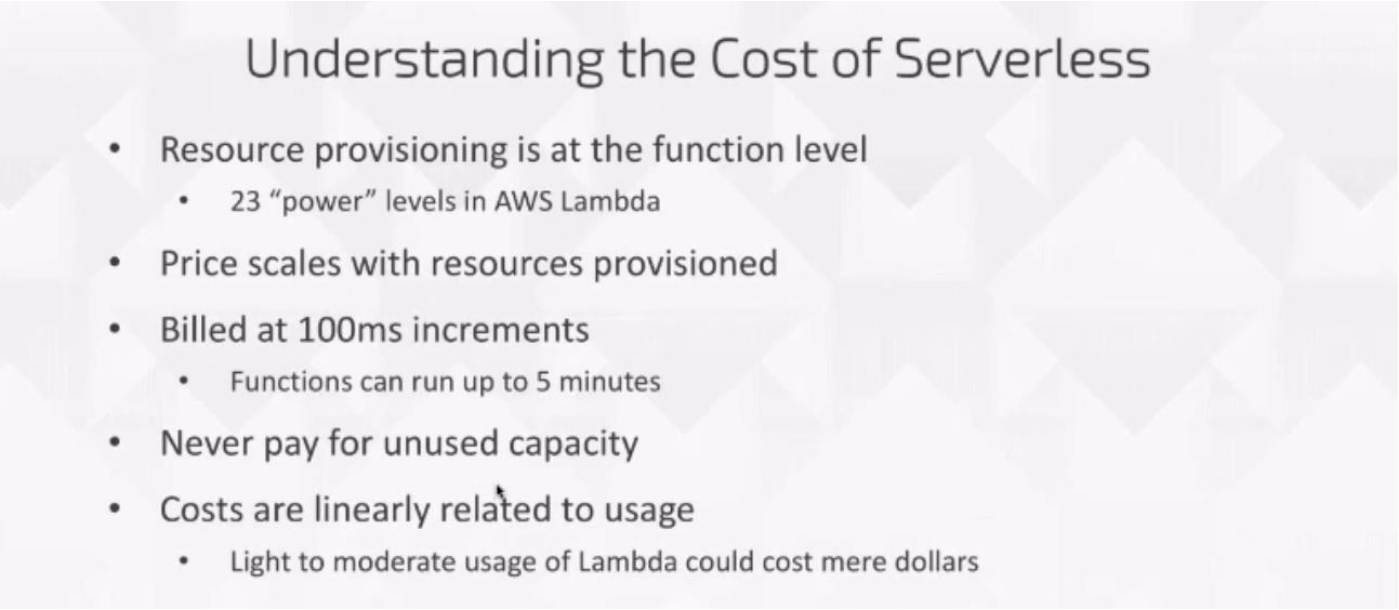
- Lambda
- API Gateway
- DynamoDB
- S3

Why Serverless? =

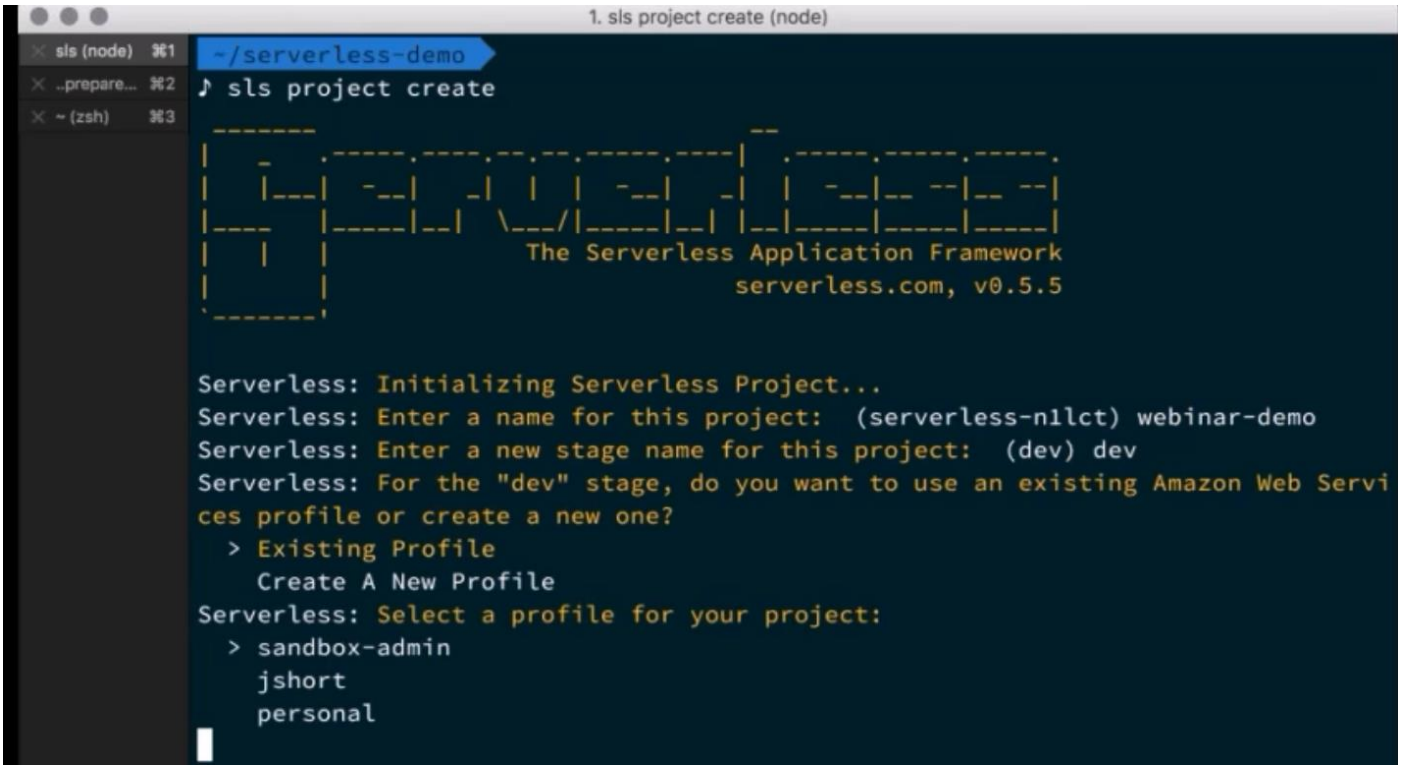
- Scalability – solved
- Fault Tolerance – solved
- No OS config
- No security patching
- Costs scale *per request*
- Most versatile platform abstraction ever

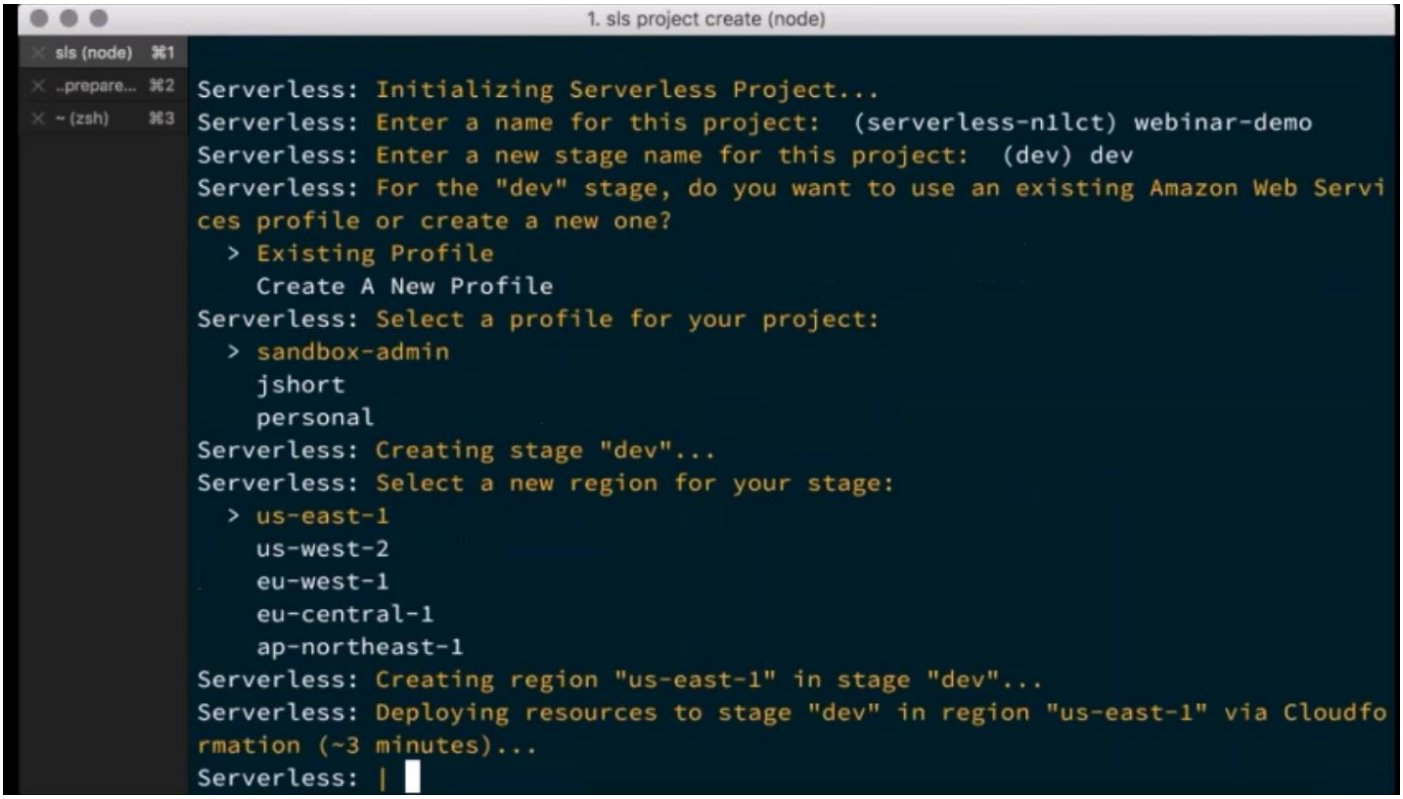
The Impact of Serverless

- Infrastructure, Configuration and Scaling are solved
- DevOps teams focus on pipelines
- Developer teams should only care about code
- "Will the app scale", becomes "how to make this function scale"
  - Serverless creates new paradigms, such as Lambda Fanout
- Use the best tool (or language) for a job, runtime provided
  - Data science? Python. Web server? Node.js. Hardcore algorithms? Java, or BYO.
- Scheduling available creates flexible cron replacement

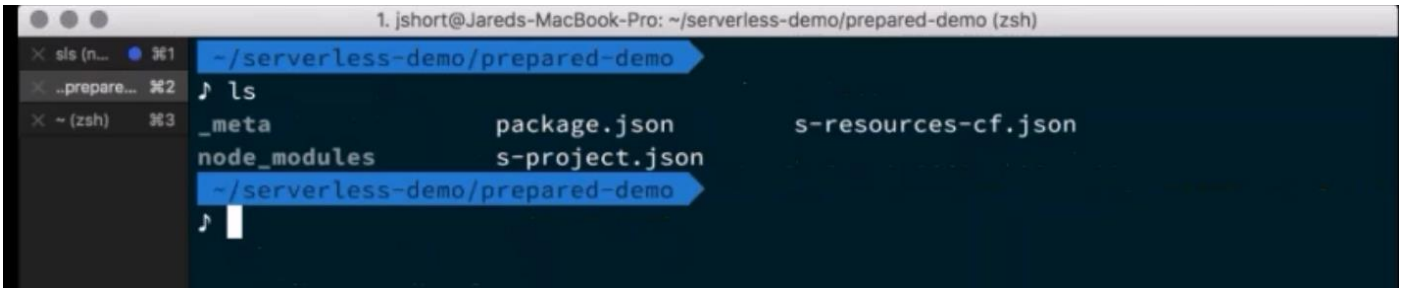


An event can be trigerred when you do something with any of the available AWS services, an event passes some JSON data about that event that you can act on like passing it to a lambda function. An event can be ***a Cron expression*** where you can define a 1 min resolution as when you want a lambda function fire off and do something like some scheduled job. S3 events are from GET/PUT/POST calls and can also be used for performing some work by subscribing a lambda function to some set of S3 events. You can also specify a prefix for a directory in your S3 bucket to act only on for the subscriptions.

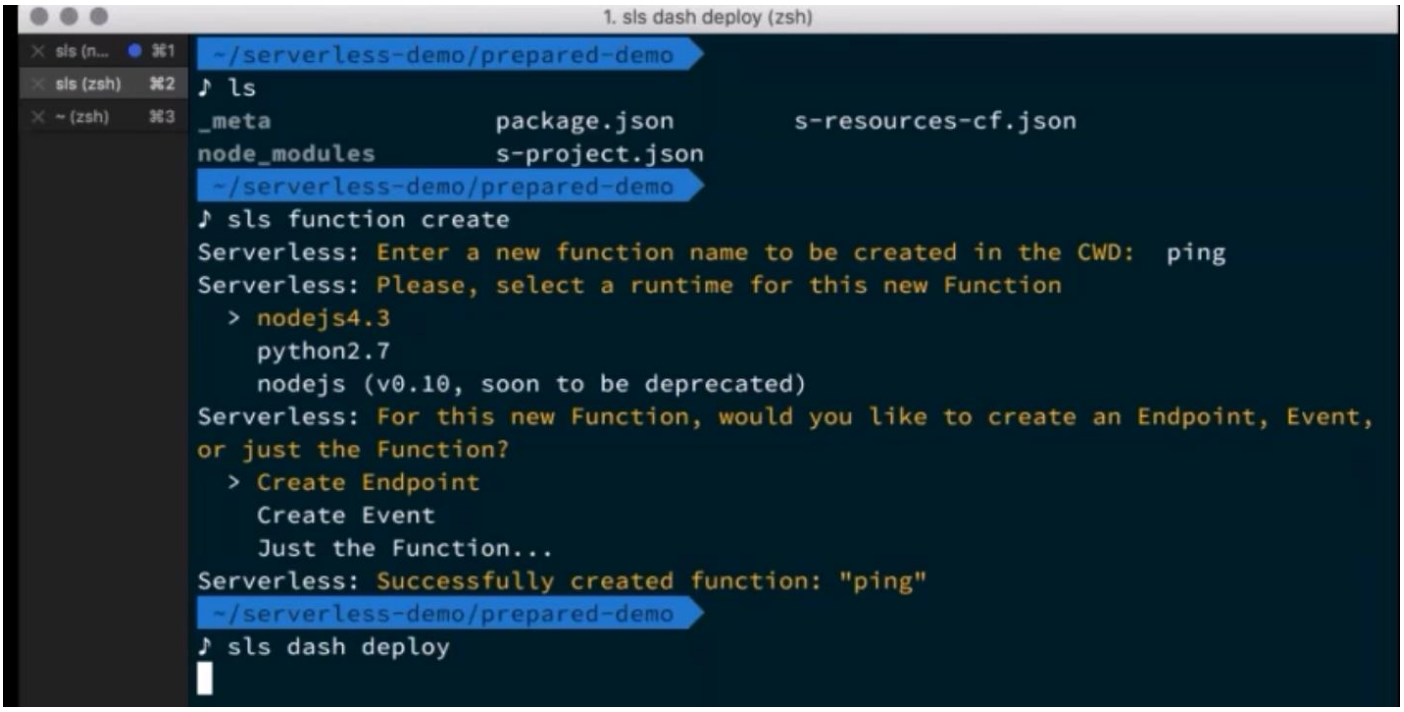




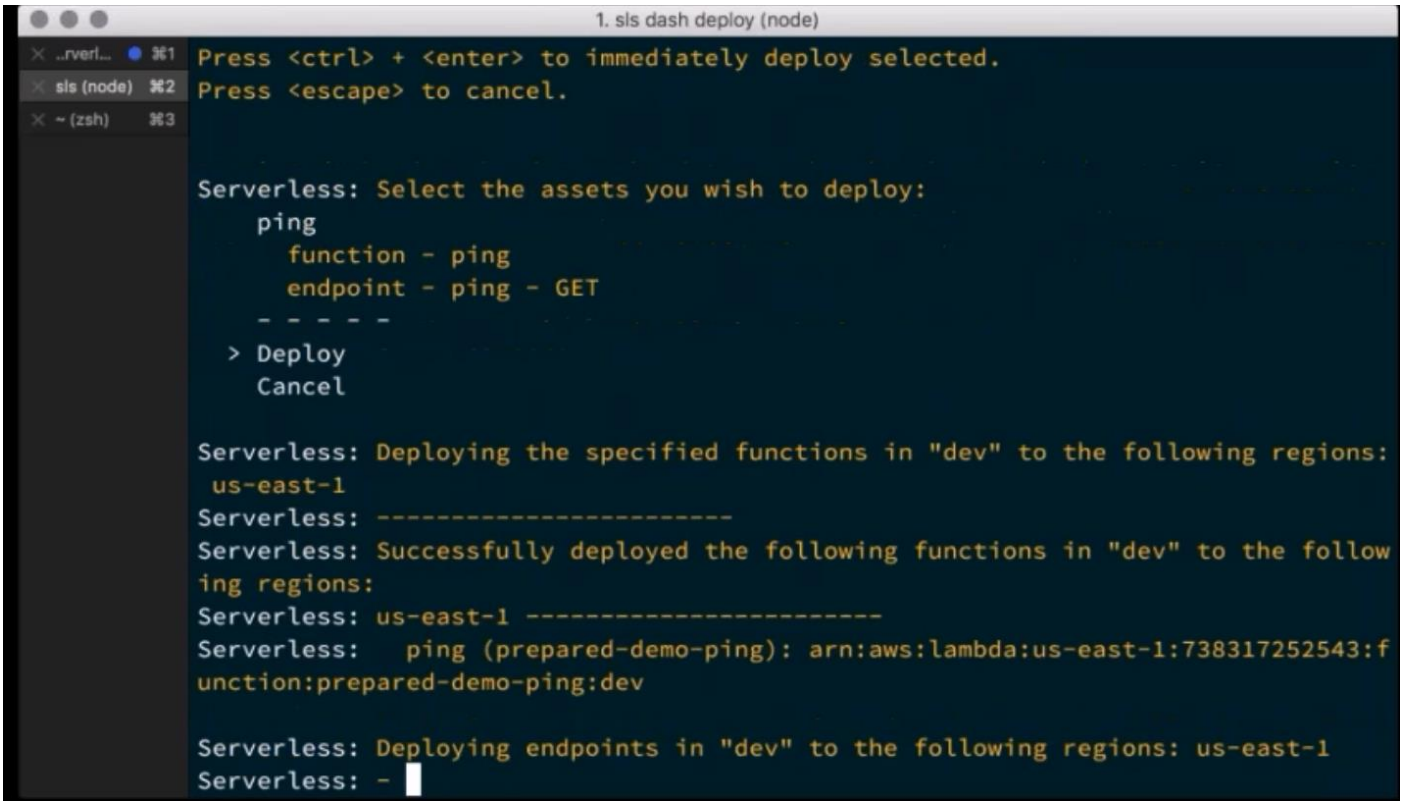
On the backend, serverless is using CloudFormation to orchestrate all the needed AWS resources for us using our account credentials.



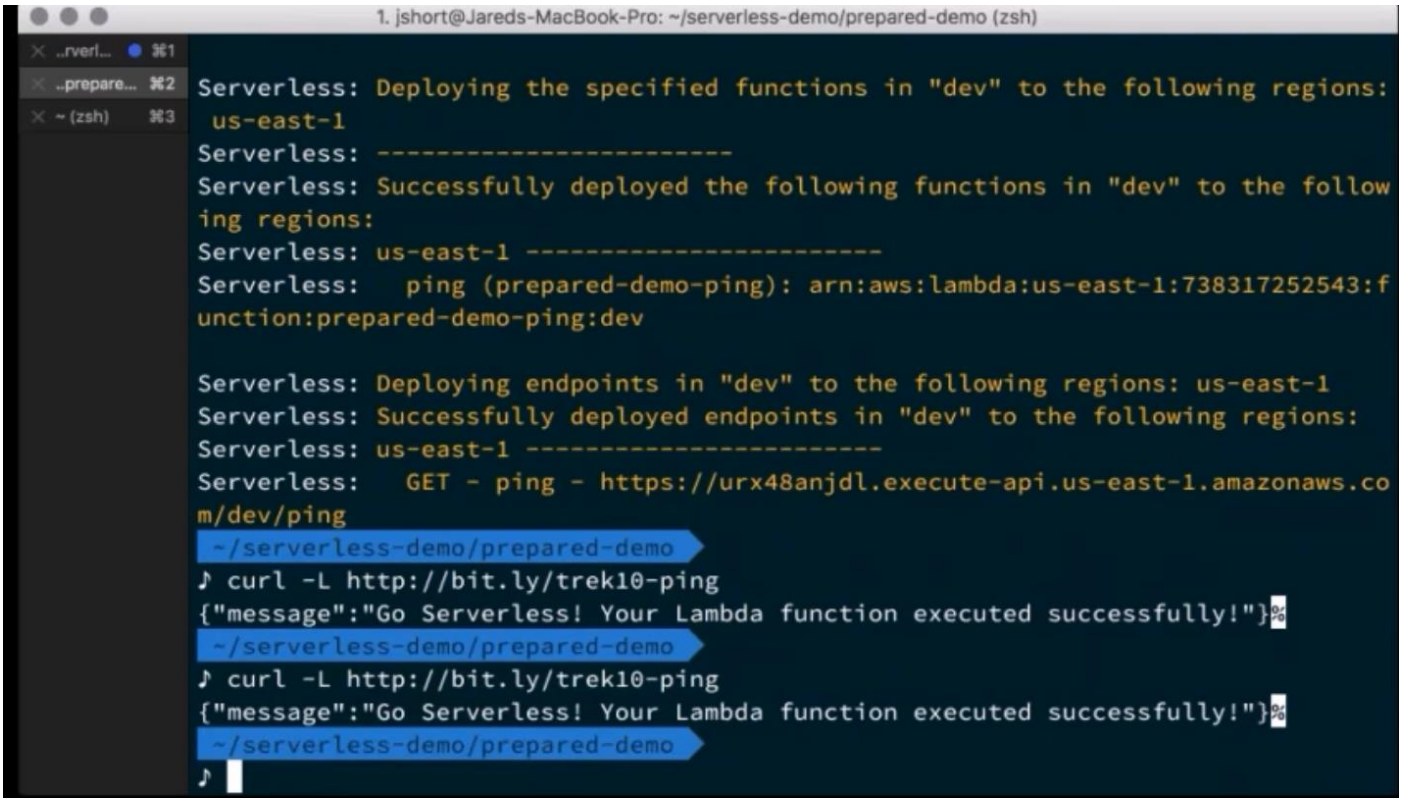
We now have a serverless project, it does not do anything yet.



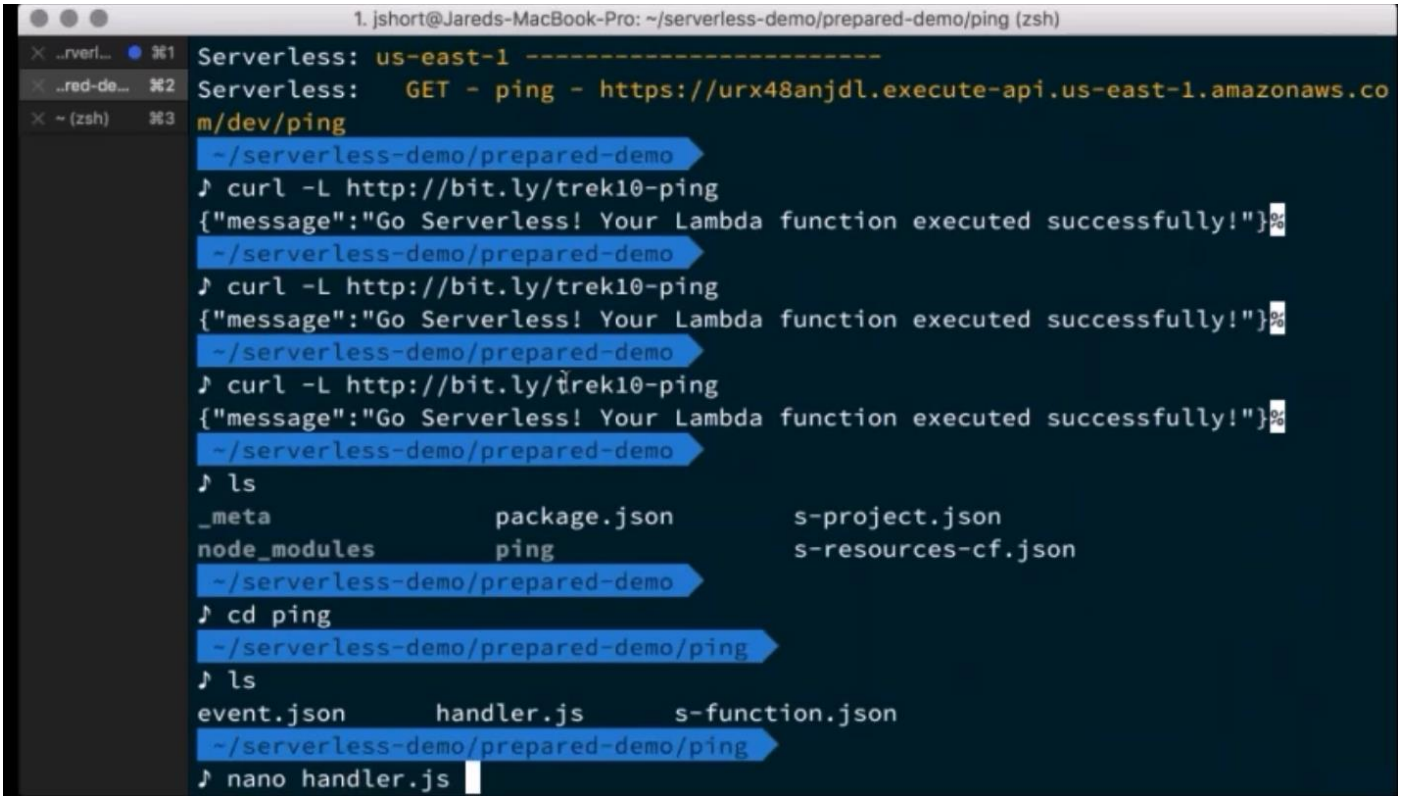
We create a lambda function in our project as above. We can also see the dashboard



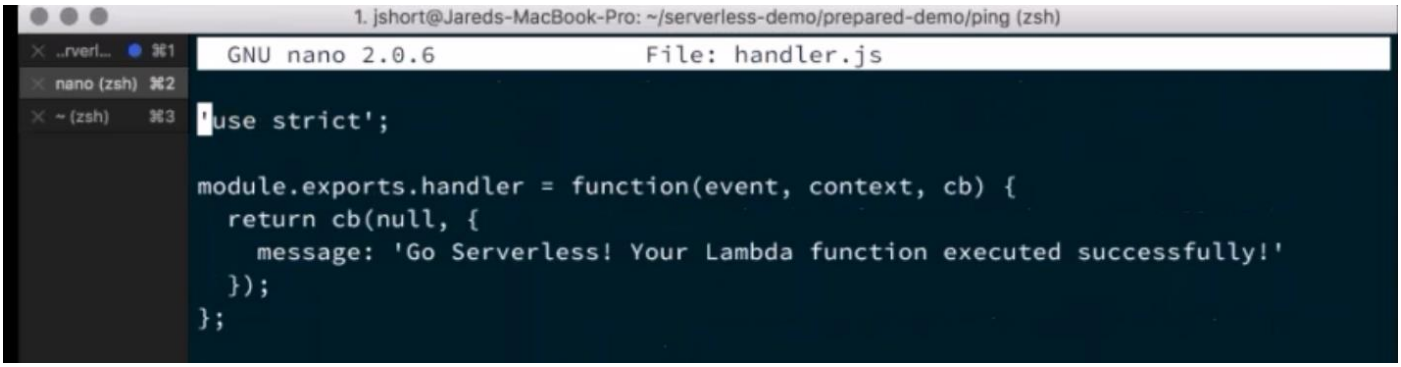
Choose deploy to deploy the function



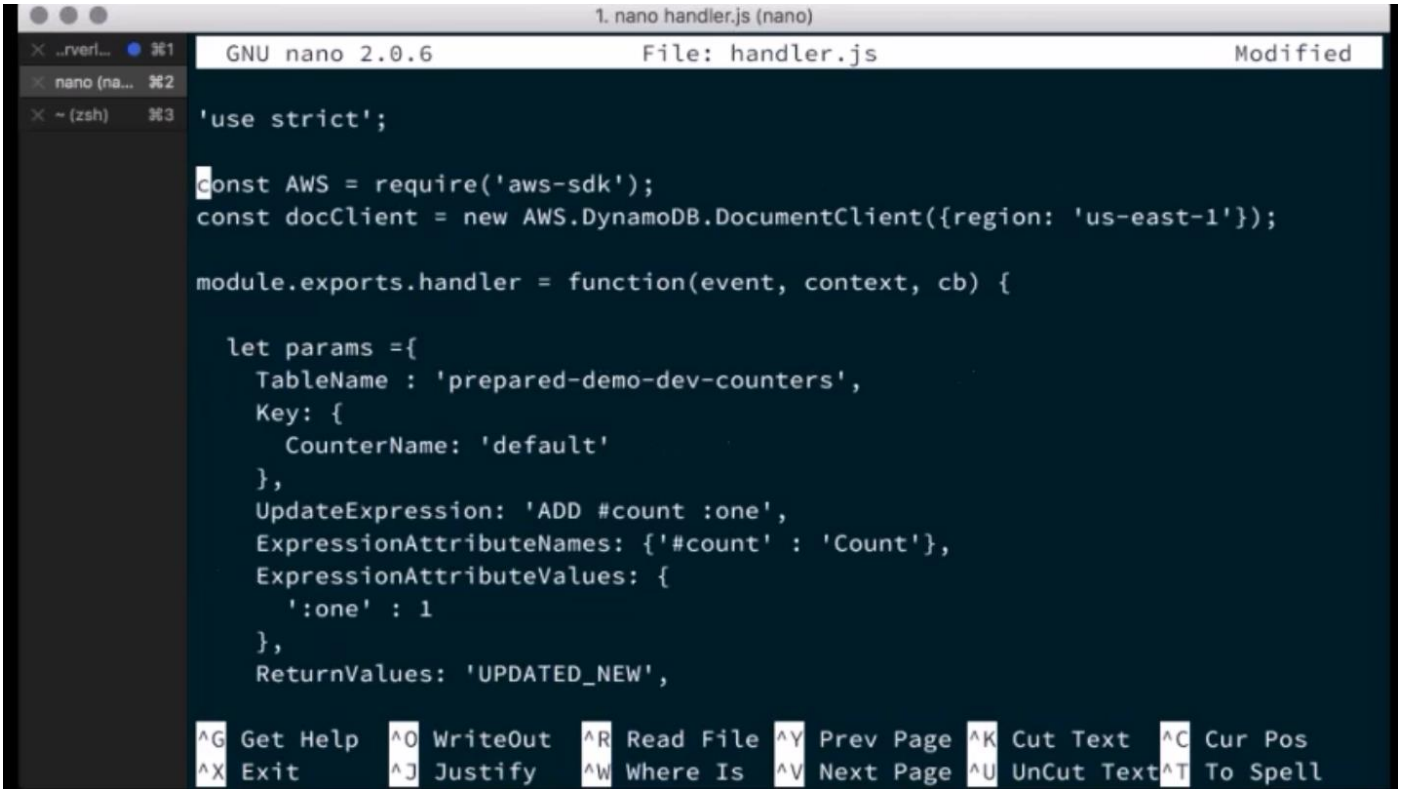
We now have our lambda function deployed and a URL that we can call as above

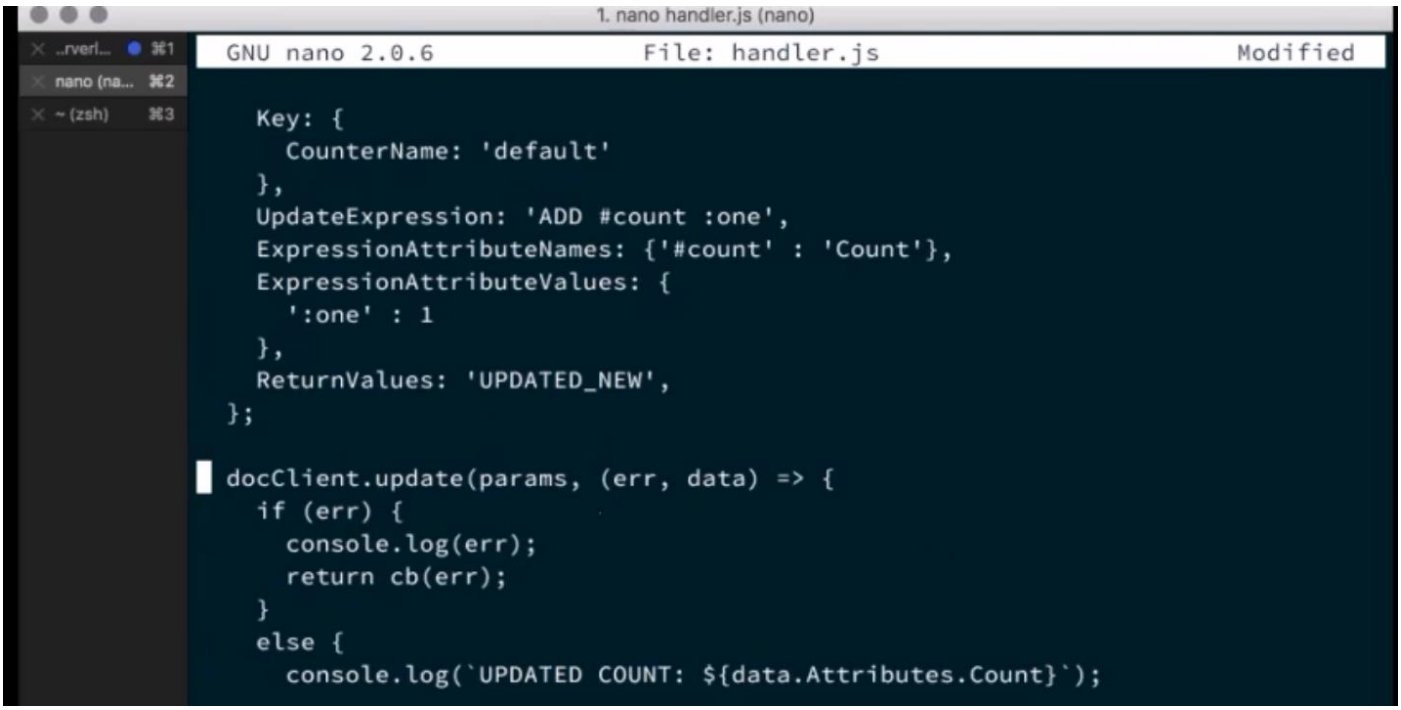


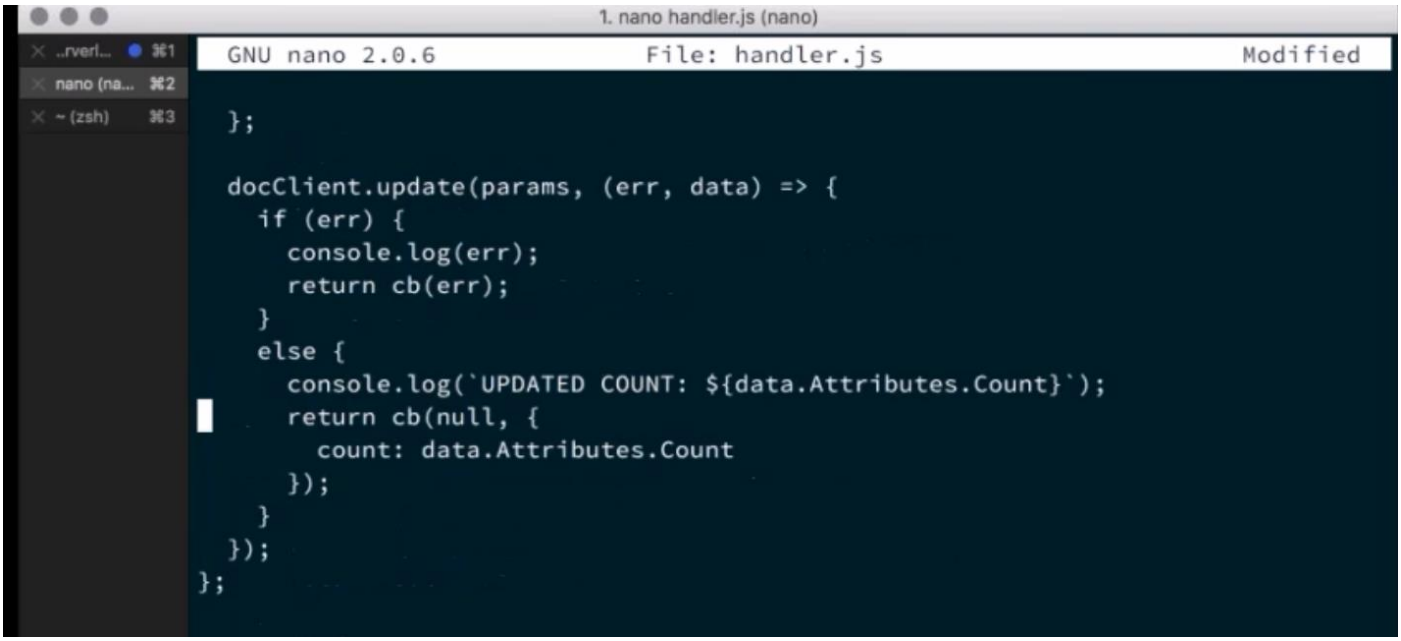
Let us do some edit here

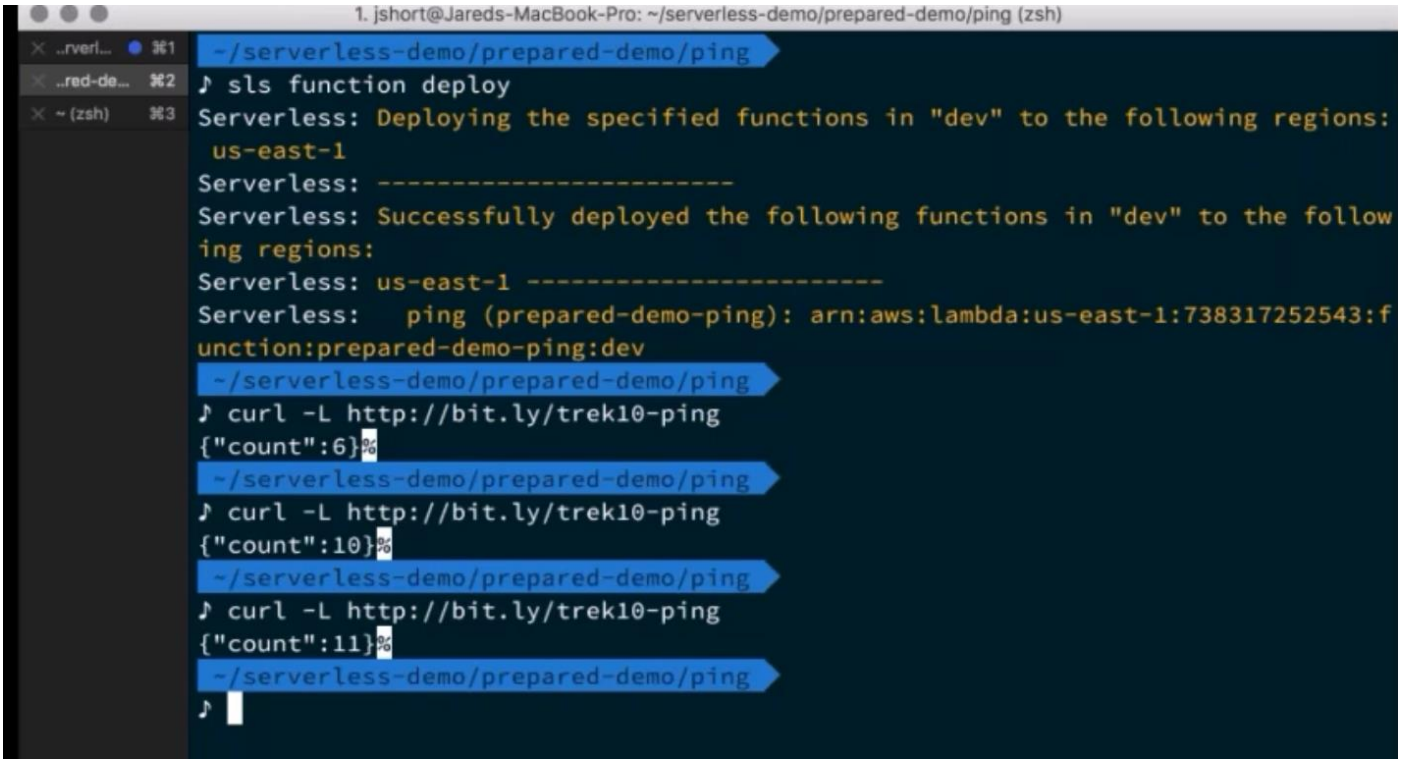


We are going to replace this code and paste in some code that uses DynamoDB on the backend with a table. We are just going to build a simple counter using a count expression as below

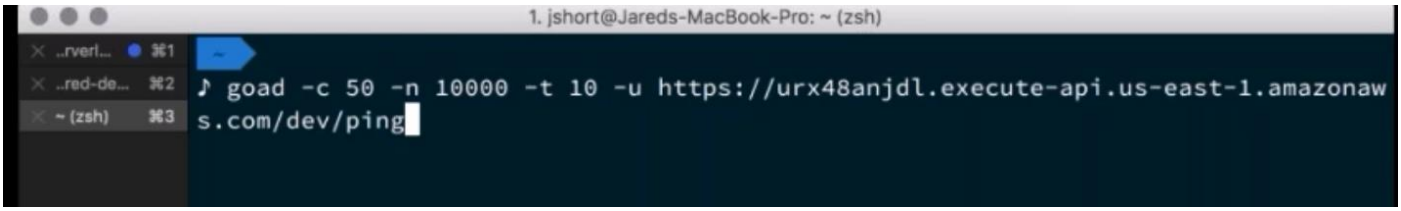

1. nano handler.js (nano)
..rverl... ⌘1
nano (na... ⌘2
~ (zsh) ⌘3
GNU nano 2.0.6 File: handler.js Modified
'use strict';
const AWS = require('aws-sdk');
const docClient = new AWS.DynamoDB.DocumentClient({region: 'us-east-1'});
module.exports.handler = function(event, context, cb) {
let params ={
TableName : 'prepared-demo-dev-counters',
Key: {
CounterName: 'default'
},
UpdateExpression: 'ADD #count :one',
ExpressionAttributeNames: {'#count' : 'Count'},
ExpressionAttributeValues: {
':one' : 1
},
ReturnValues: 'UPDATED_NEW',
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell


1. nano handler.js (nano)
..rverl... ⌘1
nano (na... ⌘2
~ (zsh) ⌘3
GNU nano 2.0.6 File: handler.js Modified
Key: {
CounterName: 'default'
},
UpdateExpression: 'ADD #count :one',
ExpressionAttributeNames: {'#count' : 'Count'},
ExpressionAttributeValues: {
':one' : 1
},
ReturnValues: 'UPDATED_NEW',
};
docClient.update(params, (err, data) => {
if (err) {
console.log(err);
return cb(err);
}
else {
console.log(`UPDATED COUNT: ${data.Attributes.Count}`);

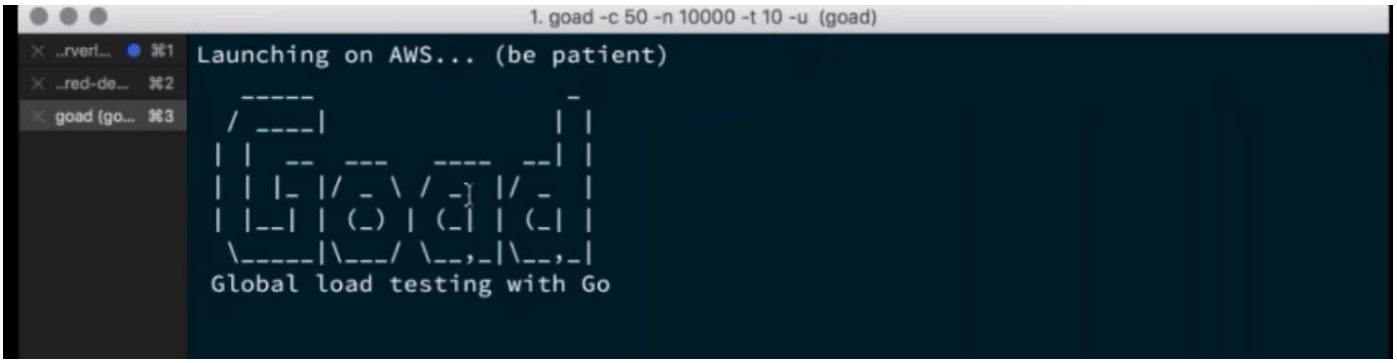


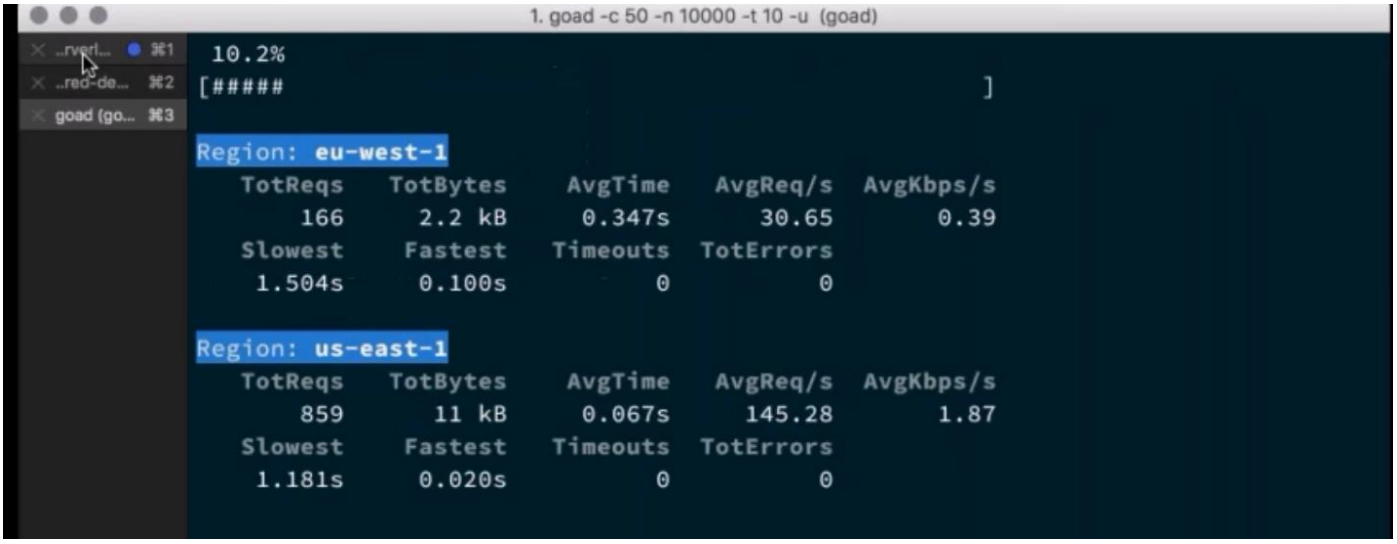


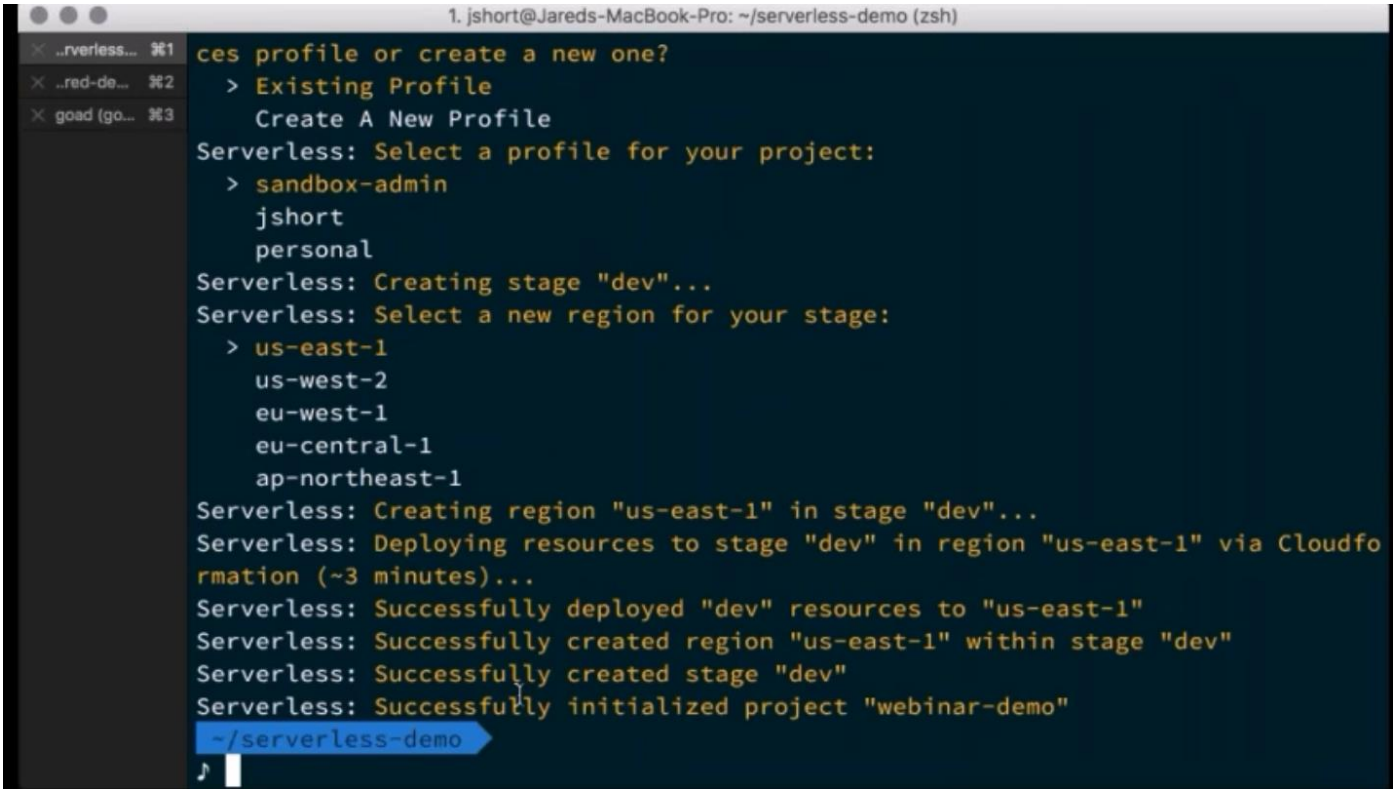
We can then use the ***sls function deploy*** command above to redeploy our lambda function

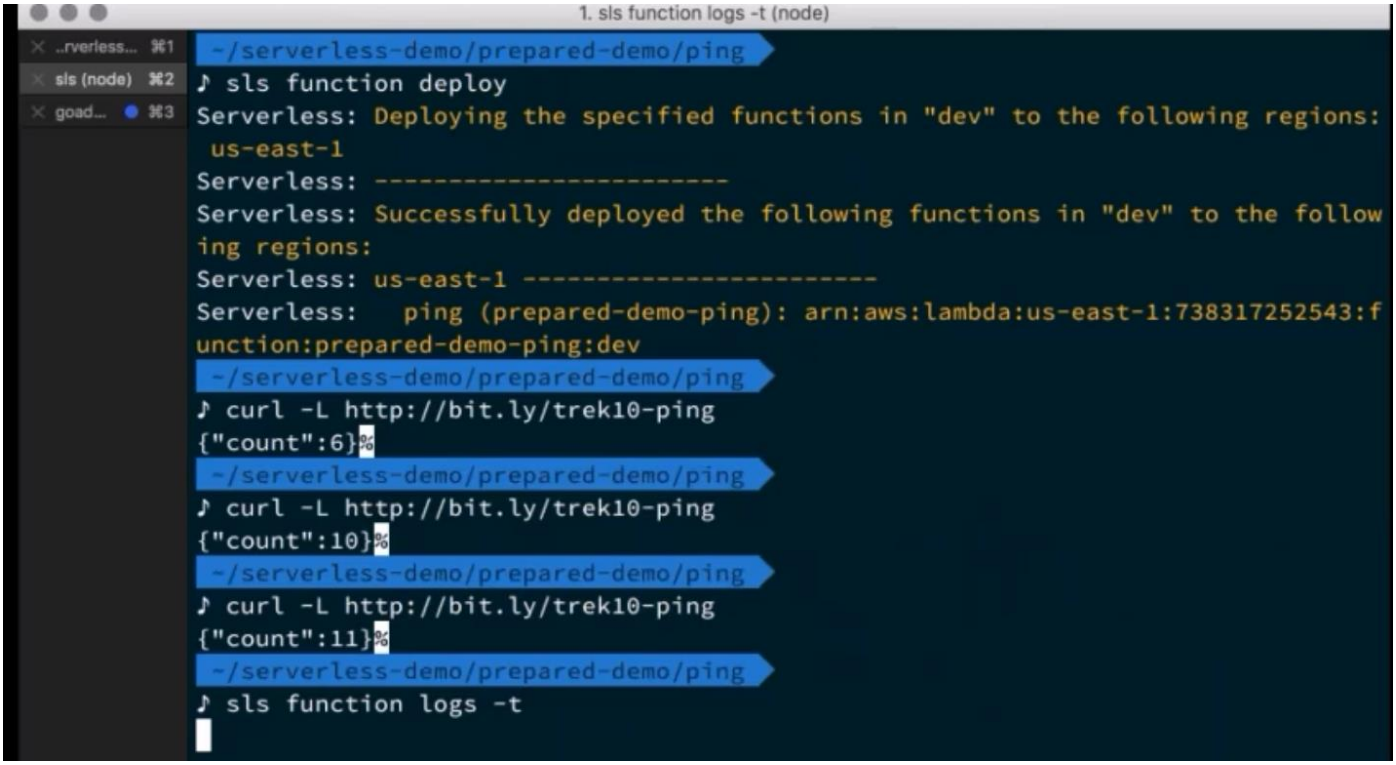


We also have a load testing tool that was also written on lambda, we are going to use this tool to test our new endpoint and see the count increase in real time. We are going to hit the endpoint with 50 concurrent requests, 10000 in total

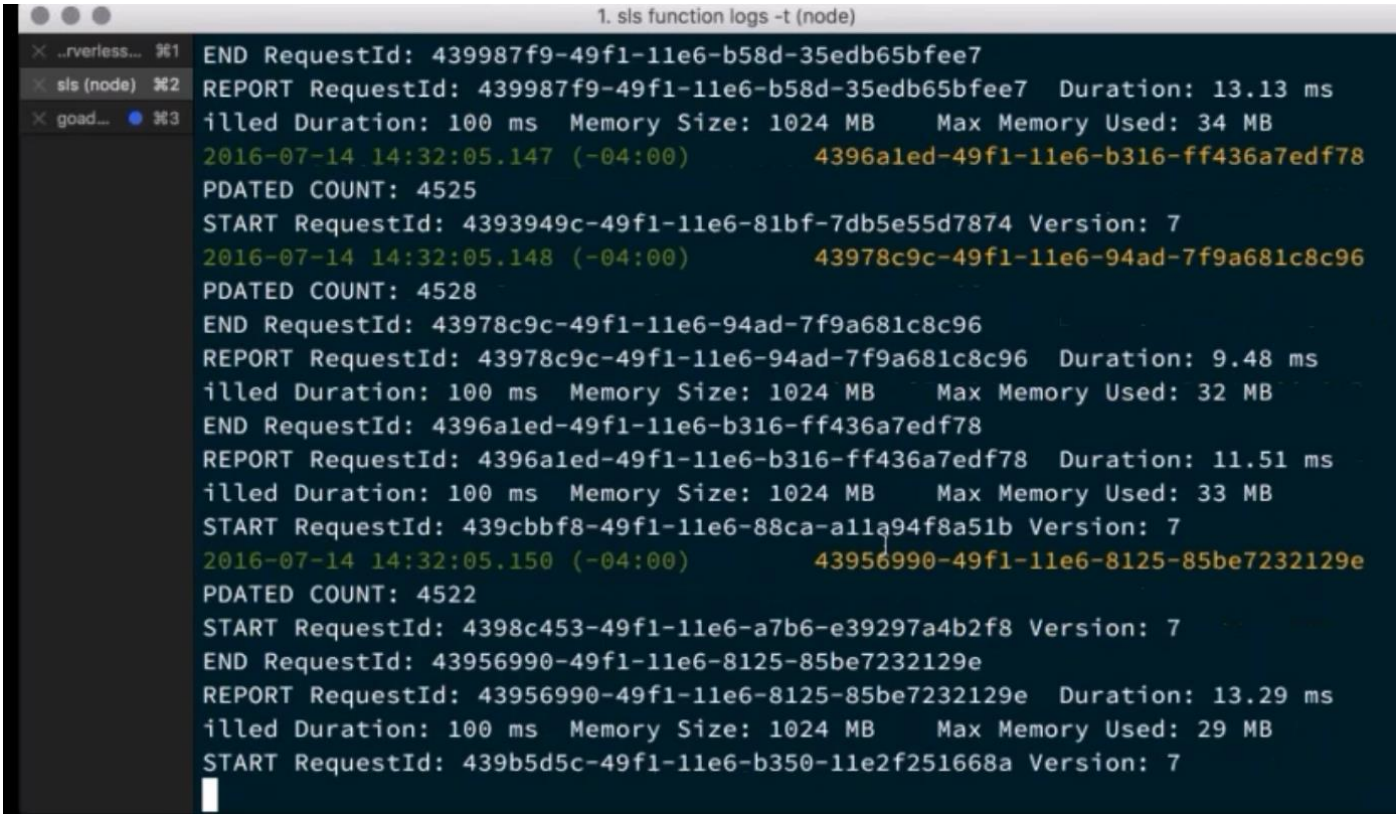

1. goad -c 50 -n 10000 -t 10 -u (goad)
..rverl... ⌘1
..red-de... ⌘2
goad (go... ⌘3
Launching on AWS... (be patient)
Global load testing with Go

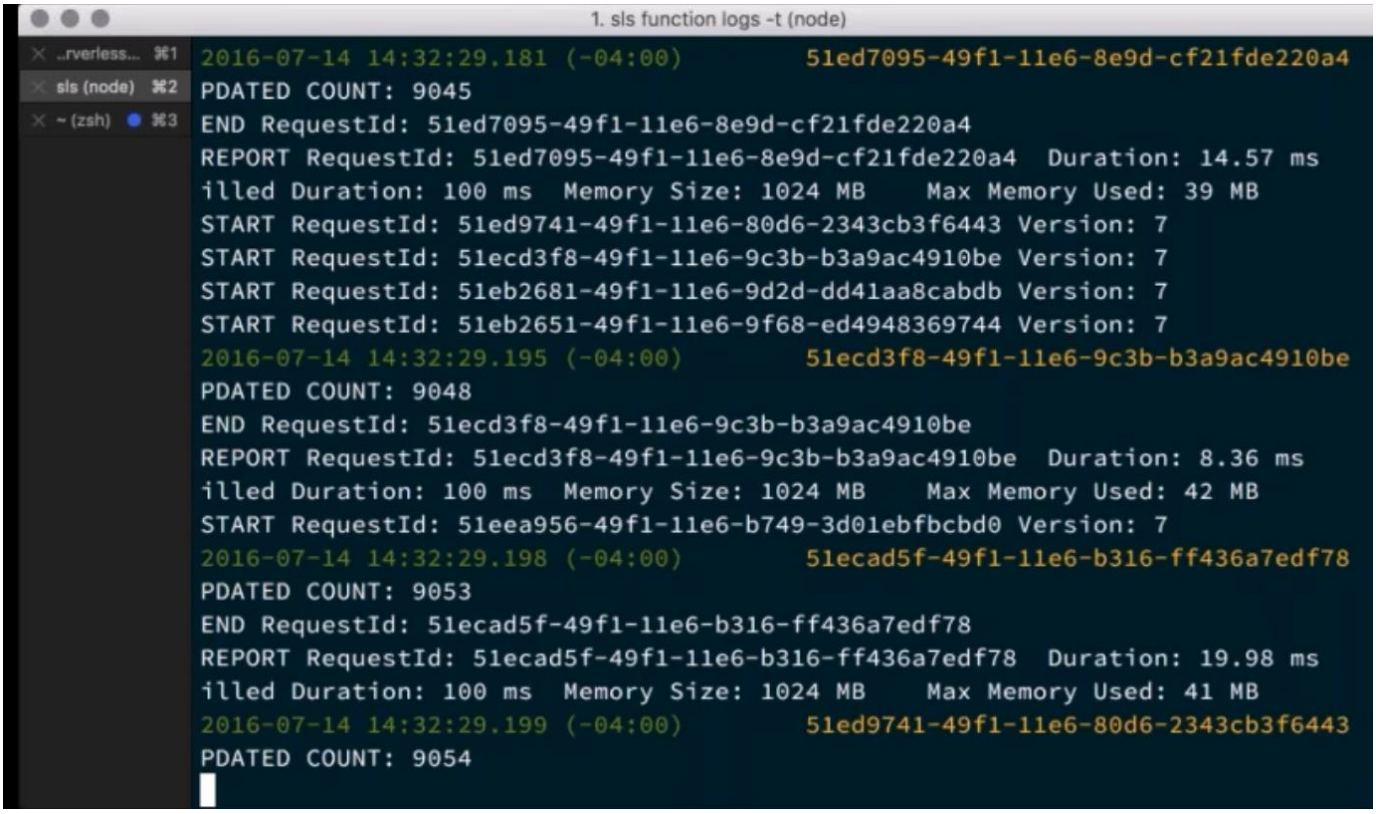

1. goad -c 50 -n 10000 -t 10 -u (goad)
..rverl... ⌘1
..red-de... ⌘2
goad (go... ⌘3
10.2%
[#####
]
Region: eu-west-1
TotReqs TotBytes AvgTime AvgReq/s AvgKbps/s
166 2.2 kB 0.347s 30.65 0.39
Slowest Fastest Timeouts TotErrors
1.504s 0.100s 0 0
Region: us-east-1
TotReqs TotBytes AvgTime AvgReq/s AvgKbps/s
859 11 kB 0.067s 145.28 1.87
Slowest Fastest Timeouts TotErrors
1.181s 0.020s 0 0


1. jshort@Jareds-MacBook-Pro: ~/serverless-demo (zsh)
..rverless... ⌘1
..red-de... ⌘2
goad (go... ⌘3
ces profile or create a new one?
> Existing Profile
Create A New Profile
Serverless: Select a profile for your project:
> sandbox-admin
jshort
personal
Serverless: Creating stage "dev"...
Serverless: Select a new region for your stage:
> us-east-1
us-west-2
eu-west-1
eu-central-1
ap-northeast-1
Serverless: Creating region "us-east-1" in stage "dev"...
Serverless: Deploying resources to stage "dev" in region "us-east-1" via Cloudfo
rmation (~3 minutes)...
Serverless: Successfully deployed "dev" resources to "us-east-1"
Serverless: Successfully created region "us-east-1" within stage "dev"
Serverless: Successfully created stage "dev"
Serverless: Successfully initialized project "webinar-demo"
~/serverless-demo

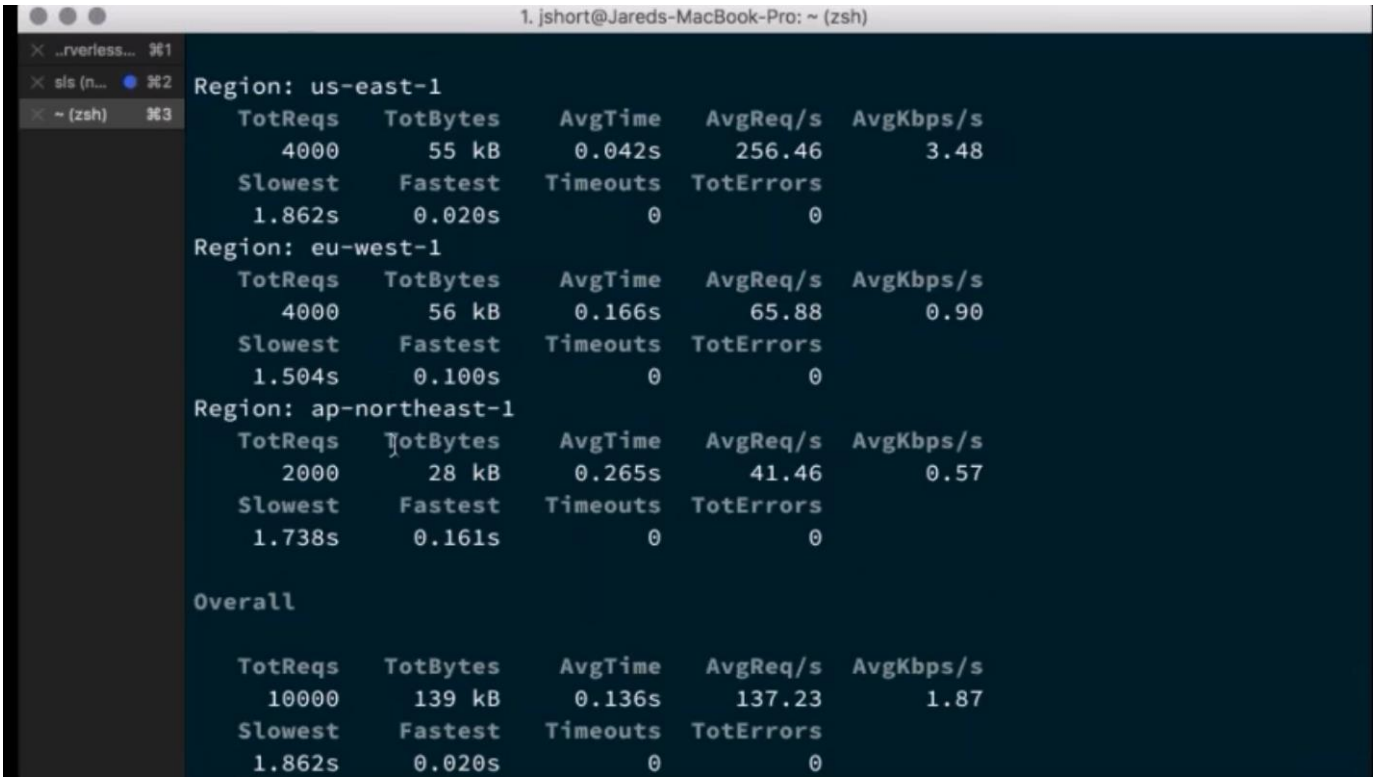


Serverless has a built-in functionality that enables us to see/tail our logs using the ***sls function logs -t*** command

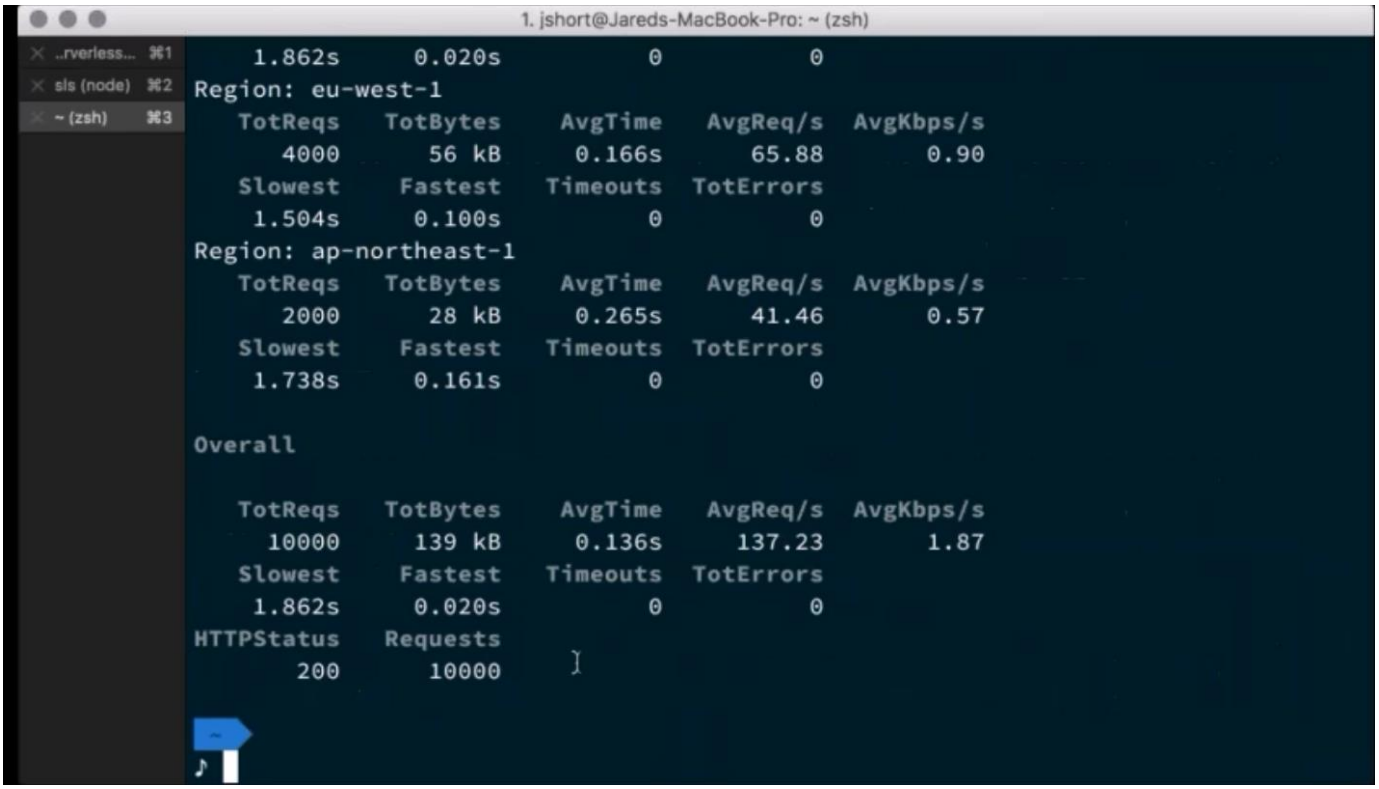




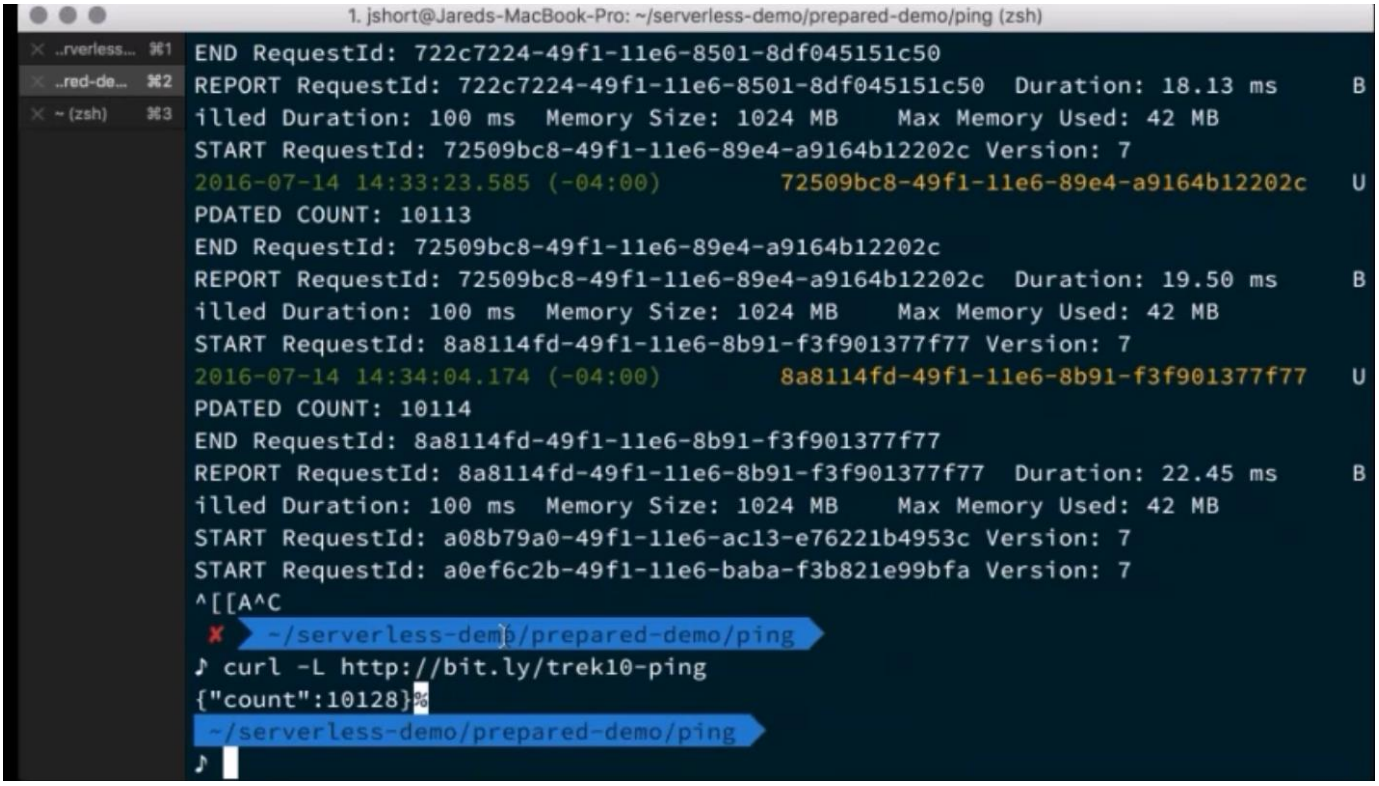
We can now start seeing the logs come through for the requests

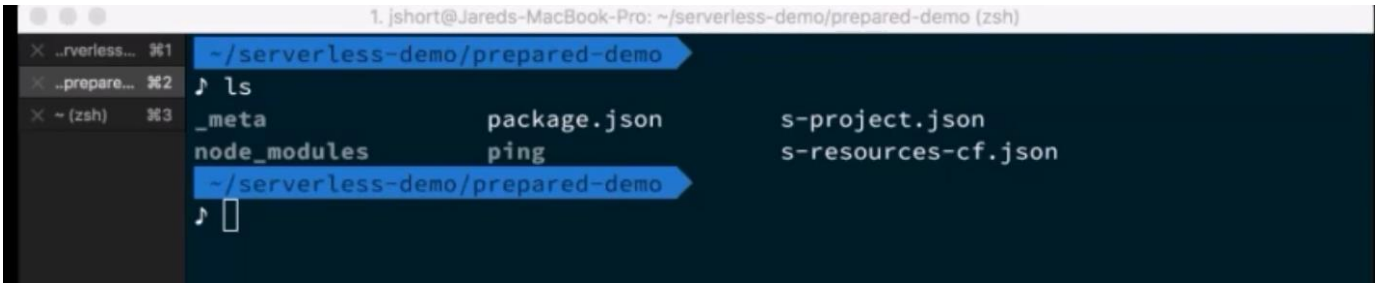


Note that the requests are also made to come from several different regions and there were no error responses from our lambda function endpoint

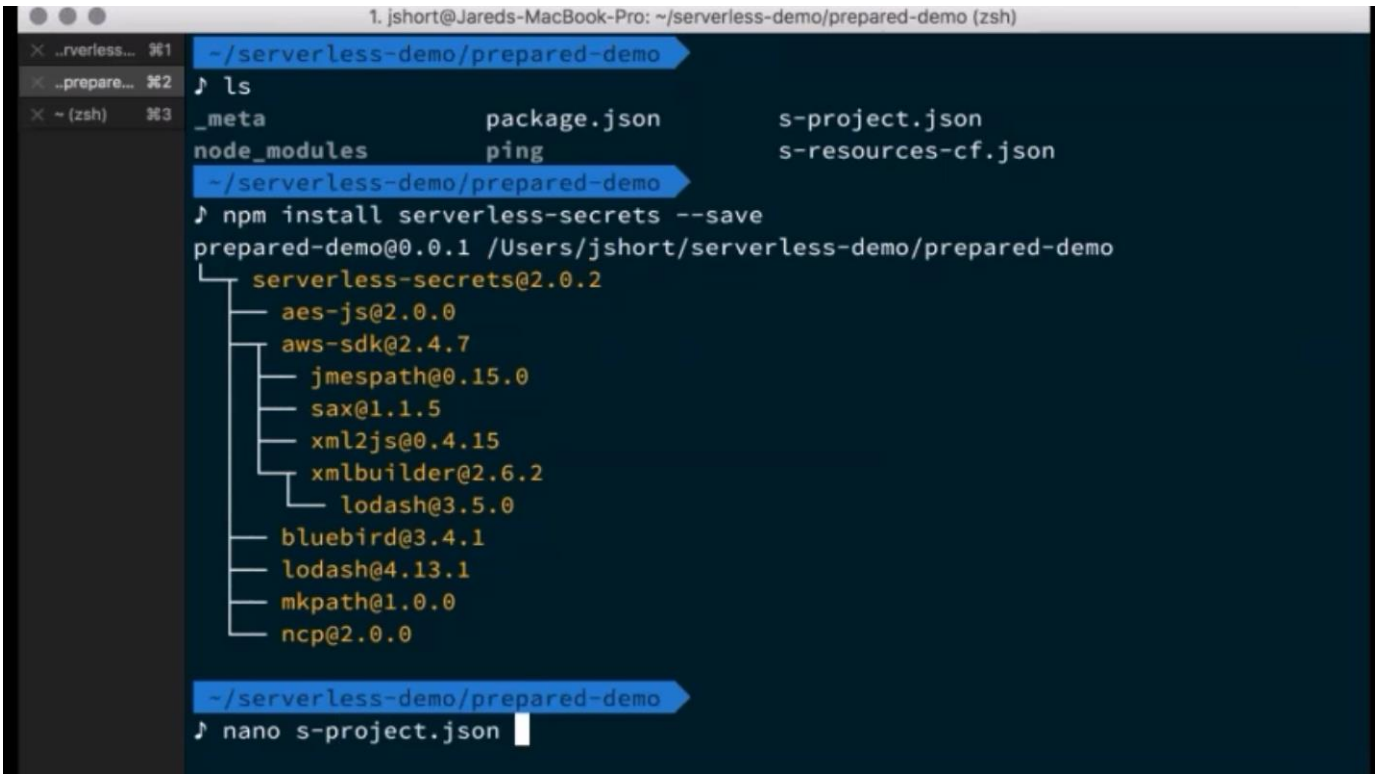


Our load test is now finished, we took 10000 request hits and had an average request time of 137.23 requests served each second from our lambda function endpoint.

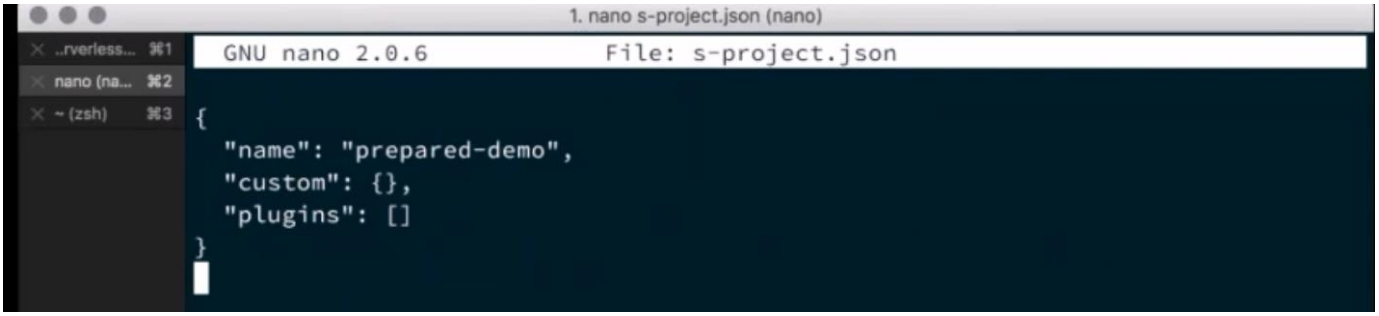




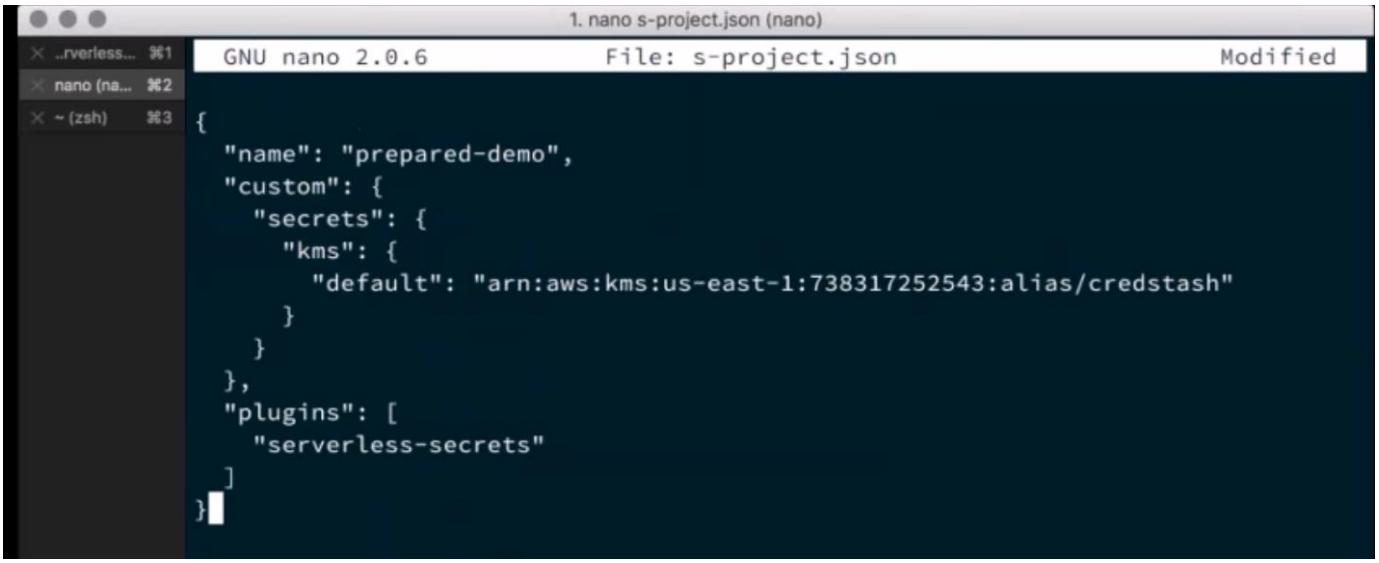
The entire serverless framework is being organized as plugins, it is built to be extremely flexible, override and extendable. You can write a secrets management plugin to manage all your secrets.



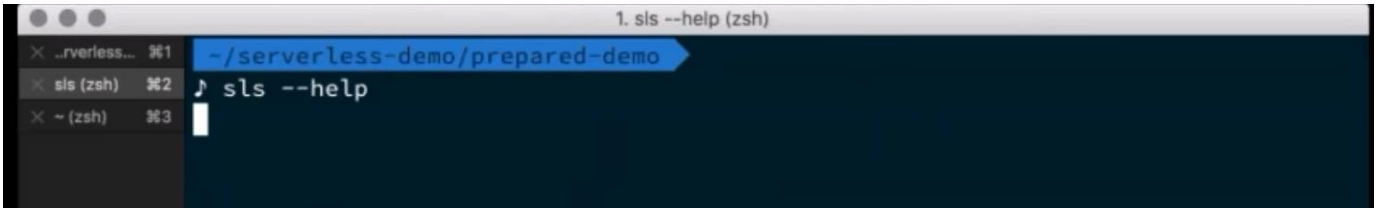
Let us install the ***serverless-secrets plugin*** to our project using the NPM ***npm install serverless-secrets --save*** command as above. You then need to let serverless know about this new plugin by editing your project configuration as below

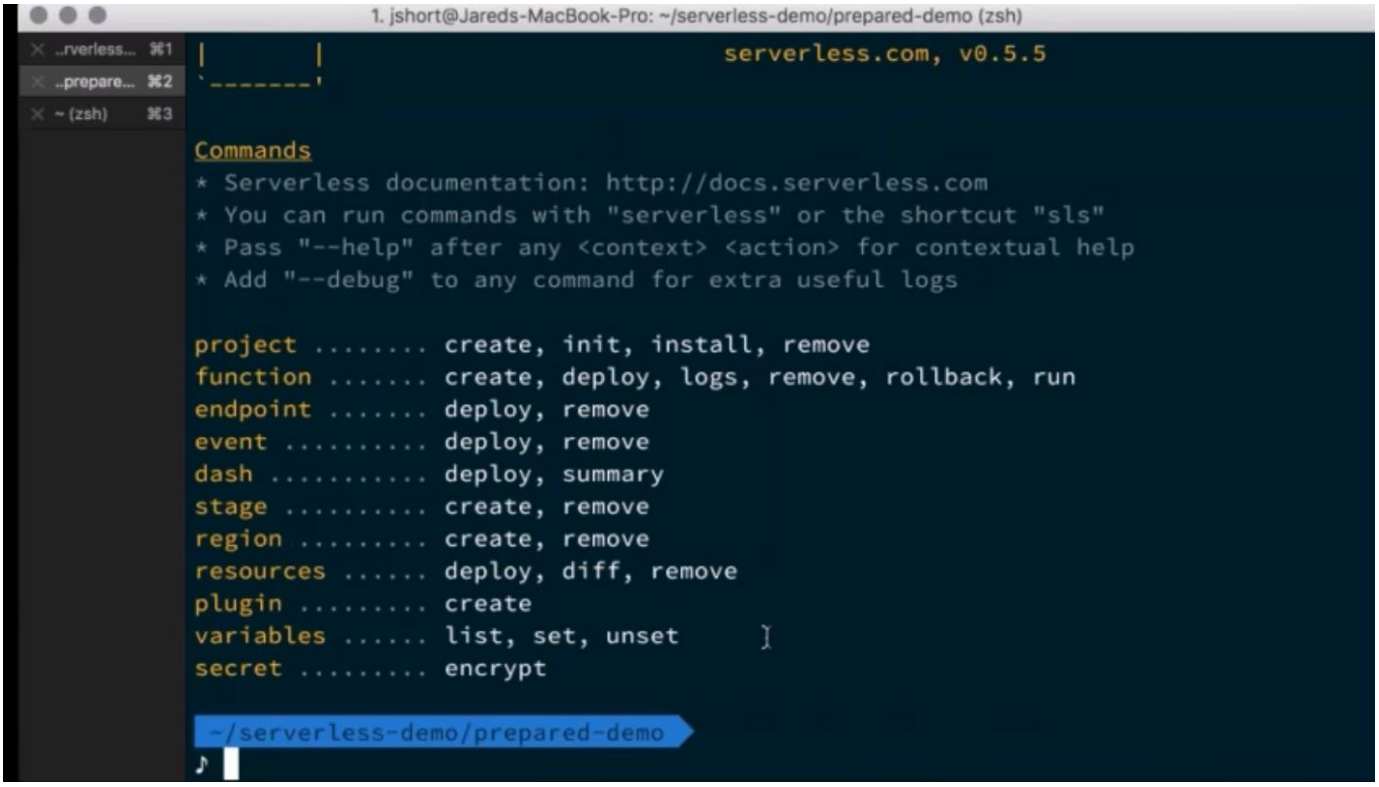


We are going to add in the code snippet below,

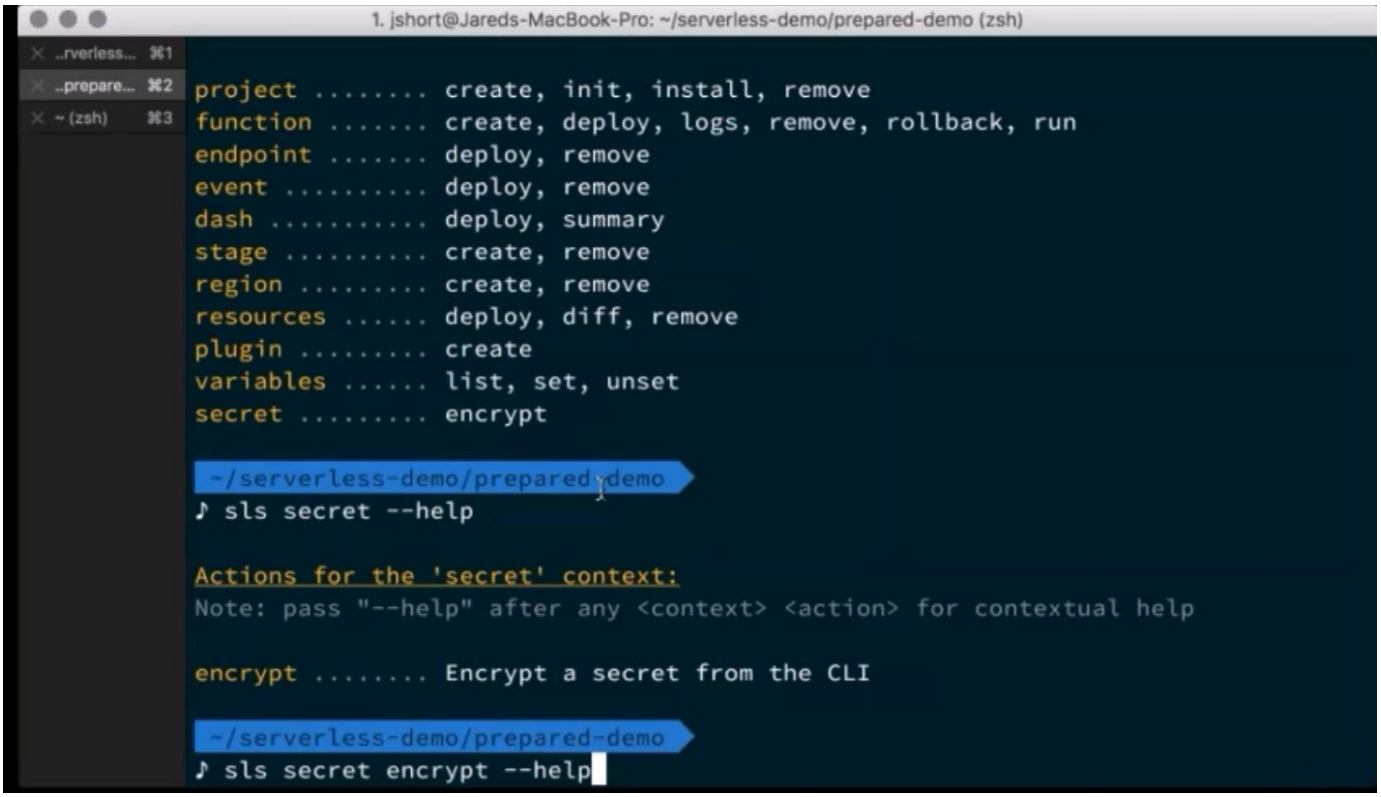


We are telling serverless about some custom configuration for some plugin called ***serverless-secrets*** in this project, we are then letting the secrets plugin know that we are going to be using the AWS Key Management Service ***KMS*** that using hardware encryption modeuls/devices to protect certificates and encrypt data for us using encryption keys.

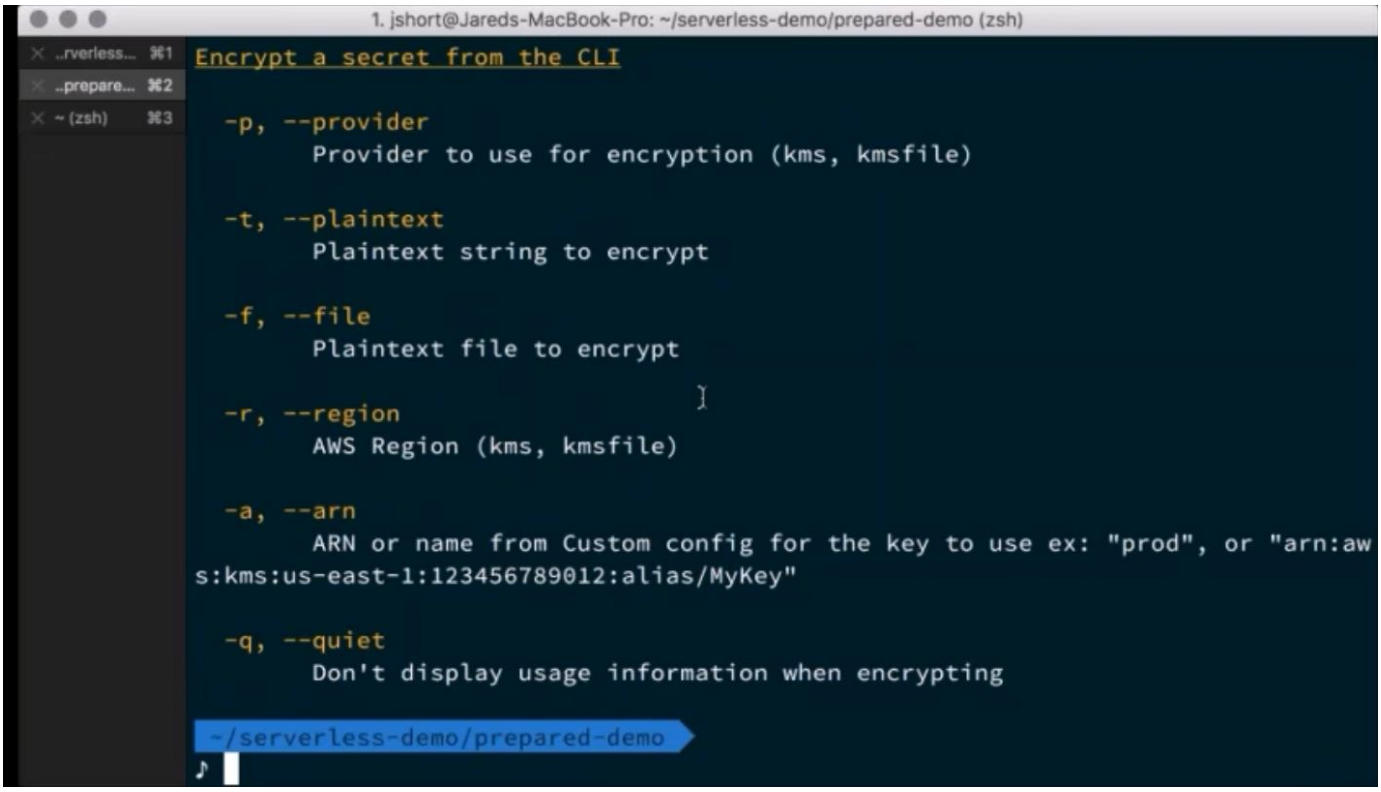




We can see that we now have an available command called ***secret***.



We have an ***encrypt*** action that we can do in the context of the ***secret*** command.

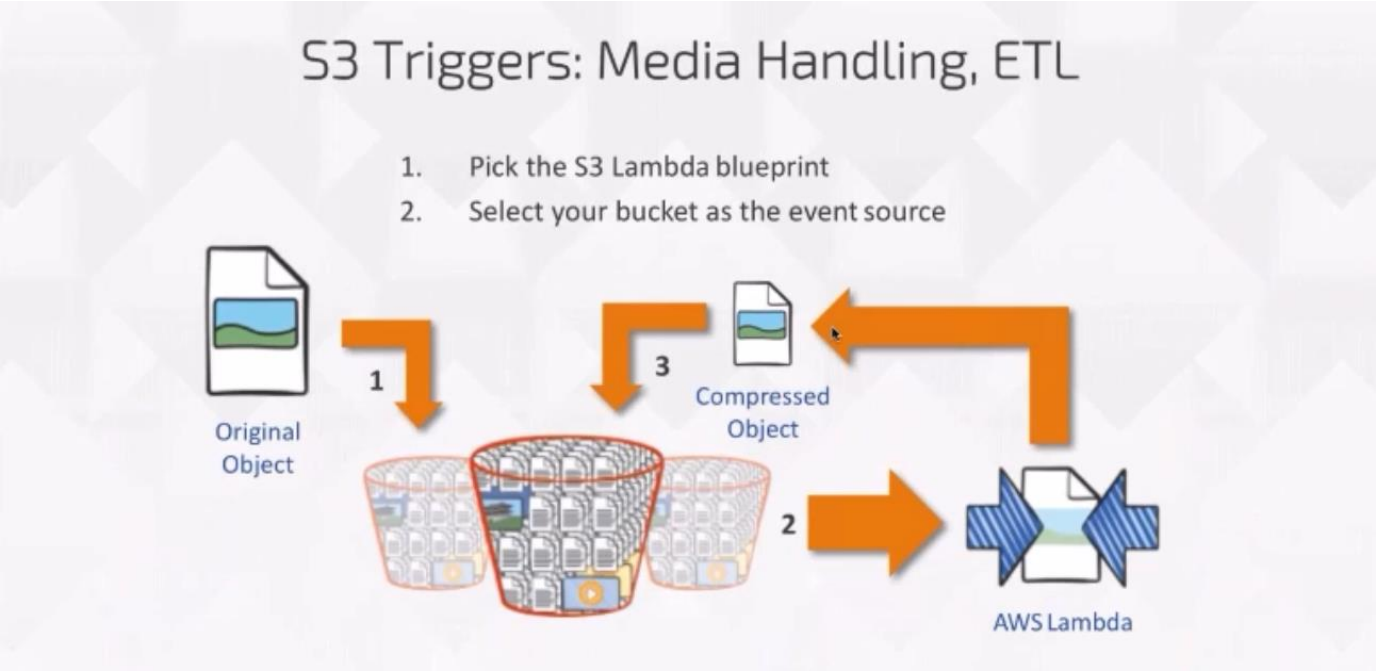


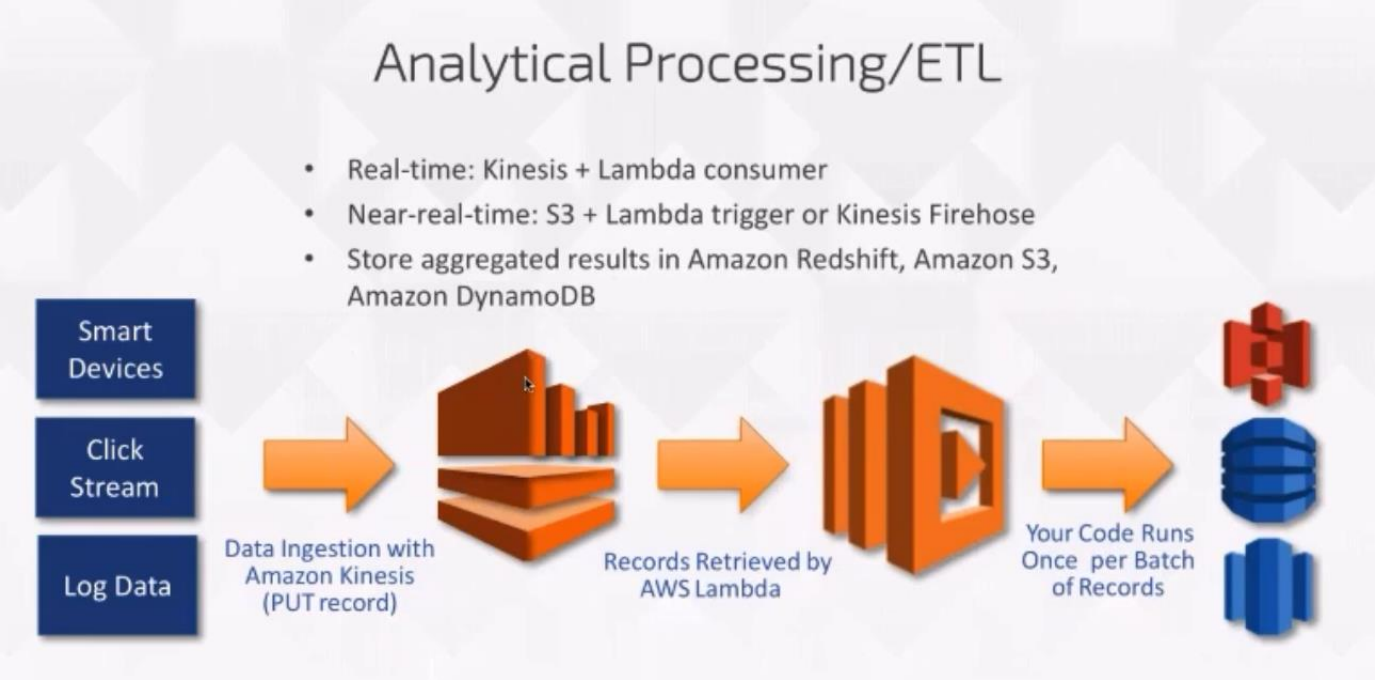
This means that we can encrypt a secret from the CLI using any of the flags shown above.

```
~/serverless-demo/prepared-demo
♪ sls secret encrypt --provider kms --plaintext superSecretStuff
Serverless: Your secret has been encypted.
            Please copy paste the string below into your _meta folder, and inclu
de it
            in the relevant s-function environment configurations.

            kms::arn:aws:kms:us-east-1:738317252543:alias/credstash::CiDZOVE8Sg9
RUIiajI+8gPaN5ChCUVn2EYTKRtpcDQ+s0RKXAQEBAgB42TlRPEoPUVCImoyPvID2jeQoQlFZ9hGEykb
aXA0PrNEAAABuMGwGCSqGSIb3DQEHBqBfMF0CAQAwWAYJKoZIhvcNAQcBMB4GCWCGSAFlAwQBLjARBAw
XxUJvnXNohjJJoaYCARCAKzfYPlBCXeeK7byNlJlz20EpnrEh3crXfKVYz2SDQi0di+N5fJeQxbTLXJM
=

~/serverless-demo/prepared-demo
♪
```

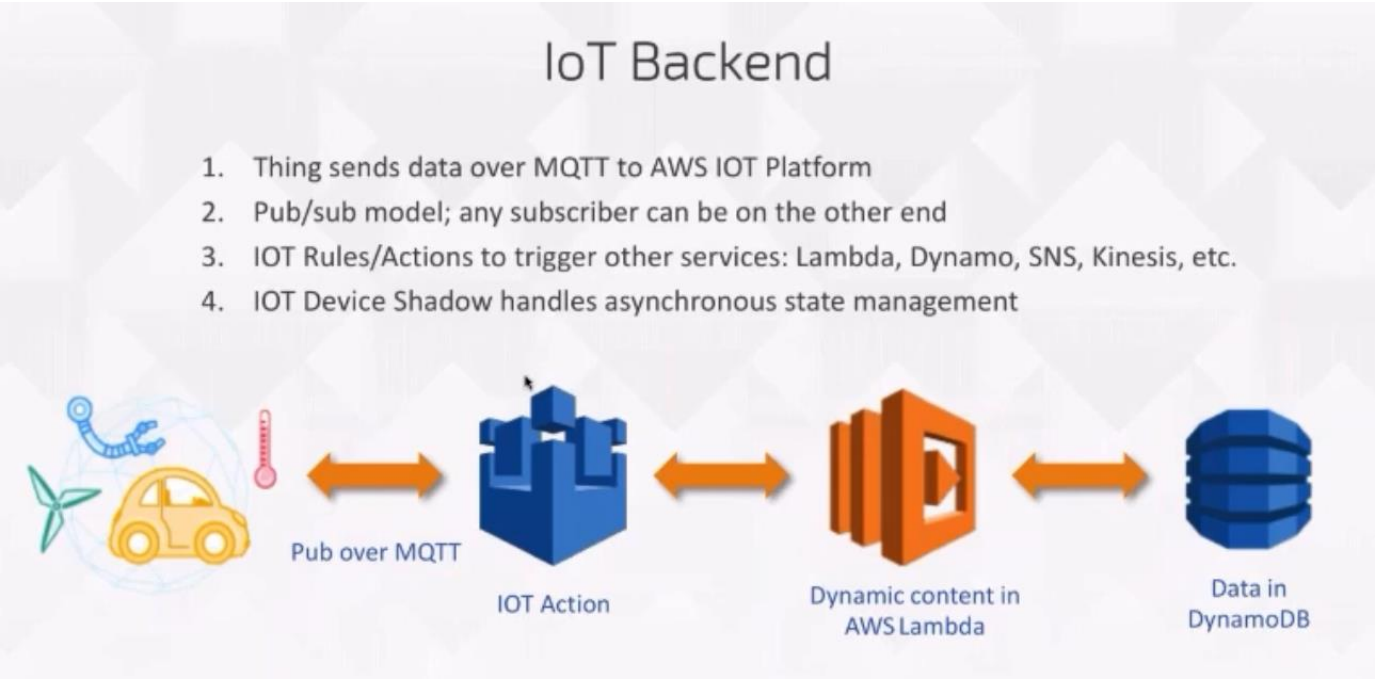
Using the ***sls secret encrypt - -provider kms - -plaintext superSecretStuff*** command, we can create an encrypted version of any plaintext secret we have, we can then use the encrypted value of the plaintext secret in the ***serverless*** environment to use.

This is easy because so many events in the AWS environment can be used to trigger a lambda function. CloudWatch cannot directly trigger a lambda but it can trigger an SNS notification service which in turn can be used to trigger a lambda function. The lambda can then update code to make the necessary change to your deployment environment.





You can put your data into Kinesis Streams and multiple data subscribers can consume the data as it comes in. Kinesis can also be used to trigger a lambda function to do things like run some logic on data as it comes in or in a batch of every 10 data points that come into the stream.



The AWS IOT Platform can be used to trigger lambda functions (lambda functions can also publish events or messages to your IOT platform also) and other AWS services, giving you the ability to build an analytics backend that uses the IOT platform to build a database, data lake, or even real-time information display.