



Common Serverless Use Cases & Architectures

Andy Warzon
Founder & CEO, Trek10

Jared Short
Director of DevOps, Trek10



Serverless? But what about...



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

Understanding the Cost of Serverless

- Resource provisioning is at the function level
 - 23 “power” levels in AWS Lambda
- Price scales with resources provisioned
- Billed at 100ms increments
 - Functions can run up to 5 minutes
- Never pay for unused capacity
- Costs are linearly related to usage
 - Light to moderate usage of Lambda could cost mere dollars

Event-driven?

Event: JSON + Functions



An event can be triggered 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.

Serverless Framework Demo


```
1. jshort@Jareds-MacBook-Pro: ~/serverless-demo/prepared-demo (zsh)
X sls (n...  161  ~/serverless-demo/prepared-demo
X ..prepare... 162  ^ ls
X ~ (zsh)    163  _meta                package.json          s-resources-cf.json
                  node_modules          s-project.json
                  ~/serverless-demo/prepared-demo
                  ^
```

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

```
1. sls dash deploy (zsh)
X sls (n...  161  ~/serverless-demo/prepared-demo
X sls (zsh)  162  ^ ls
X ~ (zsh)    163  _meta                package.json          s-resources-cf.json
                  node_modules          s-project.json
                  ~/serverless-demo/prepared-demo
                  ^ sls function create
Serverless: Enter a new function name to be created in the CWD: ping
Serverless: Please, select a runtime for this new Function
> nodejs4.3
   python2.7
   nodejs (v0.10, soon to be deprecated)
Serverless: For this new Function, would you like to create an Endpoint, Event,
or just the Function?
> Create Endpoint
   Create Event
   Just the Function...
Serverless: Successfully created function: "ping"
~/serverless-demo/prepared-demo
^ sls dash deploy
```

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

```
1. sls dash deploy (node)
Press <ctrl> + <enter> to immediately deploy selected.
Press <escape> to cancel.

Serverless: Select the assets you wish to deploy:
  ping
    function - ping
    endpoint - ping - GET
  - - - - -
> Deploy
Cancel

Serverless: Deploying the specified functions in "dev" to the following regions:
us-east-1
Serverless: -----
Serverless: Successfully deployed the following functions in "dev" to the follow
ing regions:
Serverless: us-east-1 -----
Serverless:   ping (prepared-demo-ping): arn:aws:lambda:us-east-1:738317252543:f
unction:prepared-demo-ping:dev

Serverless: Deploying endpoints in "dev" to the following regions: us-east-1
Serverless: -
```

Choose deploy to deploy the function

```
1. jshort@Jareds-MacBook-Pro: ~/serverless-demo/prepared-demo (zsh)
Serverless: Deploying the specified functions in "dev" to the following regions:
us-east-1
Serverless: -----
Serverless: Successfully deployed the following functions in "dev" to the follow
ing regions:
Serverless: us-east-1 -----
Serverless:   ping (prepared-demo-ping): arn:aws:lambda:us-east-1:738317252543:f
unction:prepared-demo-ping:dev

Serverless: Deploying endpoints in "dev" to the following regions: us-east-1
Serverless: Successfully deployed endpoints in "dev" to the following regions:
Serverless: us-east-1 -----
Serverless:   GET - ping - https://urx48anjdl.execute-api.us-east-1.amazonaws.co
m/dev/ping
~/serverless-demo/prepared-demo
$ curl -L http://bit.ly/trek10-ping
{"message":"Go Serverless! Your Lambda function executed successfully!"}%
~/serverless-demo/prepared-demo
$ curl -L http://bit.ly/trek10-ping
{"message":"Go Serverless! Your Lambda function executed successfully!"}%
~/serverless-demo/prepared-demo
$
```

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

```
1. jshort@Jareds-MacBook-Pro: ~/serverless-demo/prepared-demo/ping (zsh)
Serverless: us-east-1 -----
Serverless: GET - ping - https://urx48anjdl.execute-api.us-east-1.amazonaws.com/dev/ping
~/serverless-demo/prepared-demo
$ curl -L http://bit.ly/trek10-ping
{"message":"Go Serverless! Your Lambda function executed successfully!"}%
~/serverless-demo/prepared-demo
$ curl -L http://bit.ly/trek10-ping
{"message":"Go Serverless! Your Lambda function executed successfully!"}%
~/serverless-demo/prepared-demo
$ curl -L http://bit.ly/trek10-ping
{"message":"Go Serverless! Your Lambda function executed successfully!"}%
~/serverless-demo/prepared-demo
$ ls
_meta                package.json          s-project.json
node_modules         ping                  s-resources-cf.json
~/serverless-demo/prepared-demo
$ cd ping
~/serverless-demo/prepared-demo/ping
$ ls
event.json           handler.js            s-function.json
~/serverless-demo/prepared-demo/ping
$ nano handler.js
```

Let us do some edit here

```
1. jshort@Jareds-MacBook-Pro: ~/serverless-demo/prepared-demo/ping (zsh)
GNU nano 2.0.6          File: handler.js
'use strict';

module.exports.handler = function(event, context, cb) {
  return cb(null, {
    message: 'Go Serverless! Your Lambda function executed successfully!'
  });
};
```

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)
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)
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}`);
    }
  });
```



```
1. nano handler.js (nano)
GNU nano 2.0.6 File: handler.js Modified

};

docClient.update(params, (err, data) => {
  if (err) {
    console.log(err);
    return cb(err);
  }
  else {
    console.log(`UPDATED COUNT: ${data.Attributes.Count}`);
    return cb(null, {
      count: data.Attributes.Count
    });
  }
});
};
```

```
1. jshort@Jareds-MacBook-Pro: ~/serverless-demo/prepared-demo/ping (zsh)
~/serverless-demo/prepared-demo/ping
$ sls function deploy
Serverless: Deploying the specified functions in "dev" to the following regions:
us-east-1
Serverless: -----
Serverless: Successfully deployed the following functions in "dev" to the following regions:
Serverless: us-east-1 -----
Serverless: ping (prepared-demo-ping): arn:aws:lambda:us-east-1:738317252543:function:prepared-demo-ping:dev
~/serverless-demo/prepared-demo/ping
$ curl -L http://bit.ly/trek10-ping
{"count":6}%
~/serverless-demo/prepared-demo/ping
$ curl -L http://bit.ly/trek10-ping
{"count":10}%
~/serverless-demo/prepared-demo/ping
$ curl -L http://bit.ly/trek10-ping
{"count":11}%
~/serverless-demo/prepared-demo/ping
$
```

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

```
1. jshort@Jareds-MacBook-Pro: ~ (zsh)
$ goad -c 50 -n 10000 -t 10 -u https://urx48anjdl.execute-api.us-east-1.amazonaws.com/dev/ping
```

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

[illegible]

```

1. goad -c 50 -n 10000 -t 10 -u (goad)

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... 第1 ces profile or create a new one?
< ..red-de... 第2 > Existing Profile
< goad (go... 第3 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
```

```
1. sls function logs -t (node)
X ..rverless...  361  ~/serverless-demo/prepared-demo/ping
X sls (node)  362  ↵ sls function deploy
X goad...  363  Serverless: Deploying the specified functions in "dev" to the following regions:
us-east-1
Serverless: -----
Serverless: Successfully deployed the following functions in "dev" to the follow
ing regions:
Serverless: us-east-1 -----
Serverless:   ping (prepared-demo-ping): arn:aws:lambda:us-east-1:738317252543:f
unction:prepared-demo-ping:dev
~/serverless-demo/prepared-demo/ping
↵ curl -L http://bit.ly/trek10-ping
{"count":6}%
~/serverless-demo/prepared-demo/ping
↵ curl -L http://bit.ly/trek10-ping
{"count":10}%
~/serverless-demo/prepared-demo/ping
↵ curl -L http://bit.ly/trek10-ping
{"count":11}%
~/serverless-demo/prepared-demo/ping
↵ sls function logs -t
```

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

```
1. sls function logs -t (node)
X ..rverless...  361  END RequestId: 439987f9-49f1-11e6-b58d-35edb65bfee7
X sls (node)  362  REPORT RequestId: 439987f9-49f1-11e6-b58d-35edb65bfee7  Duration: 13.13 ms
X goad...  363  illed Duration: 100 ms  Memory Size: 1024 MB  Max Memory Used: 34 MB
2016-07-14 14:32:05.147 (-04:00) 4396aled-49f1-11e6-b316-ff436a7edf78
PDATED COUNT: 4525
START RequestId: 4393949c-49f1-11e6-81bf-7db5e55d7874 Version: 7
2016-07-14 14:32:05.148 (-04:00) 43978c9c-49f1-11e6-94ad-7f9a681c8c96
PDATED COUNT: 4528
END RequestId: 43978c9c-49f1-11e6-94ad-7f9a681c8c96
REPORT RequestId: 43978c9c-49f1-11e6-94ad-7f9a681c8c96  Duration: 9.48 ms
illed Duration: 100 ms  Memory Size: 1024 MB  Max Memory Used: 32 MB
END RequestId: 4396aled-49f1-11e6-b316-ff436a7edf78
REPORT RequestId: 4396aled-49f1-11e6-b316-ff436a7edf78  Duration: 11.51 ms
illed Duration: 100 ms  Memory Size: 1024 MB  Max Memory Used: 33 MB
START RequestId: 439cbbf8-49f1-11e6-88ca-a11a94f8a51b Version: 7
2016-07-14 14:32:05.150 (-04:00) 43956990-49f1-11e6-8125-85be7232129e
PDATED COUNT: 4522
START RequestId: 4398c453-49f1-11e6-a7b6-e39297a4b2f8 Version: 7
END RequestId: 43956990-49f1-11e6-8125-85be7232129e
REPORT RequestId: 43956990-49f1-11e6-8125-85be7232129e  Duration: 13.29 ms
illed Duration: 100 ms  Memory Size: 1024 MB  Max Memory Used: 29 MB
START RequestId: 439b5d5c-49f1-11e6-b350-11e2f251668a Version: 7
```



```
1. sls function logs -t (node)
X ..rverless... 361 2016-07-14 14:32:29.181 (-04:00) 51ed7095-49f1-11e6-8e9d-cf21fde220a4
X sls (node) 362 PDATED COUNT: 9045
X ~ (zsh) 363 END RequestId: 51ed7095-49f1-11e6-8e9d-cf21fde220a4
REPORT RequestId: 51ed7095-49f1-11e6-8e9d-cf21fde220a4 Duration: 14.57 ms
illed Duration: 100 ms Memory Size: 1024 MB Max Memory Used: 39 MB
START RequestId: 51ed9741-49f1-11e6-80d6-2343cb3f6443 Version: 7
START RequestId: 51ecd3f8-49f1-11e6-9c3b-b3a9ac4910be Version: 7
START RequestId: 51eb2681-49f1-11e6-9d2d-dd41aa8cabdb Version: 7
START RequestId: 51eb2651-49f1-11e6-9f68-ed4948369744 Version: 7
2016-07-14 14:32:29.195 (-04:00) 51ecd3f8-49f1-11e6-9c3b-b3a9ac4910be
PDATED COUNT: 9048
END RequestId: 51ecd3f8-49f1-11e6-9c3b-b3a9ac4910be
REPORT RequestId: 51ecd3f8-49f1-11e6-9c3b-b3a9ac4910be Duration: 8.36 ms
illed Duration: 100 ms Memory Size: 1024 MB Max Memory Used: 42 MB
START RequestId: 51eea956-49f1-11e6-b749-3d01ebfbcdb0 Version: 7
2016-07-14 14:32:29.198 (-04:00) 51ecad5f-49f1-11e6-b316-ff436a7edf78
PDATED COUNT: 9053
END RequestId: 51ecad5f-49f1-11e6-b316-ff436a7edf78
REPORT RequestId: 51ecad5f-49f1-11e6-b316-ff436a7edf78 Duration: 19.98 ms
illed Duration: 100 ms Memory Size: 1024 MB Max Memory Used: 41 MB
2016-07-14 14:32:29.199 (-04:00) 51ed9741-49f1-11e6-80d6-2343cb3f6443
PDATED COUNT: 9054
```

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

```
1. jshort@Jareds-MacBook-Pro: ~ (zsh)
X ..rverless... 361
X sls (n... 362
X ~ (zsh) 363
Region: us-east-1
  TotReqs  TotBytes  AvgTime  AvgReq/s  AvgKbps/s
    4000    55 kB    0.042s    256.46     3.48
  Slowest  Fastest  Timeouts TotErrors
    1.862s  0.020s      0         0
Region: eu-west-1
  TotReqs  TotBytes  AvgTime  AvgReq/s  AvgKbps/s
    4000    56 kB    0.166s    65.88     0.90
  Slowest  Fastest  Timeouts TotErrors
    1.504s  0.100s      0         0
Region: ap-northeast-1
  TotReqs  TotBytes  AvgTime  AvgReq/s  AvgKbps/s
    2000    28 kB    0.265s    41.46     0.57
  Slowest  Fastest  Timeouts TotErrors
    1.738s  0.161s      0         0
Overall
  TotReqs  TotBytes  AvgTime  AvgReq/s  AvgKbps/s
   10000   139 kB    0.136s   137.23     1.87
  Slowest  Fastest  Timeouts TotErrors
    1.862s  0.020s      0         0
```

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


```
1. jshort@Jareds-MacBook-Pro: ~ (zsh)
X ..rverless... %1 1.862s 0.020s 0 0
X ..sls (node) %2 Region: eu-west-1
X ~ (zsh) %3 TotReqs TotBytes AvgTime AvgReq/s AvgKbps/s
4000 56 kB 0.166s 65.88 0.90
Slowest Fastest Timeouts TotErrors
1.504s 0.100s 0 0
Region: ap-northeast-1
TotReqs TotBytes AvgTime AvgReq/s AvgKbps/s
2000 28 kB 0.265s 41.46 0.57
Slowest Fastest Timeouts TotErrors
1.738s 0.161s 0 0

Overall
TotReqs TotBytes AvgTime AvgReq/s AvgKbps/s
10000 139 kB 0.136s 137.23 1.87
Slowest Fastest Timeouts TotErrors
1.862s 0.020s 0 0
HTTPStatus Requests
200 10000
```

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.

```
1. jshort@Jareds-MacBook-Pro: ~/serverless-demo/prepared-demo/ping (zsh)
X ..rverless... %1 END RequestId: 722c7224-49f1-11e6-8501-8df045151c50
X ..red-de... %2 REPORT RequestId: 722c7224-49f1-11e6-8501-8df045151c50 Duration: 18.13 ms B
X ~ (zsh) %3 illed Duration: 100 ms Memory Size: 1024 MB Max Memory Used: 42 MB
START RequestId: 72509bc8-49f1-11e6-89e4-a9164b12202c Version: 7
2016-07-14 14:33:23.585 (-04:00) 72509bc8-49f1-11e6-89e4-a9164b12202c U
PDATED COUNT: 10113
END RequestId: 72509bc8-49f1-11e6-89e4-a9164b12202c
REPORT RequestId: 72509bc8-49f1-11e6-89e4-a9164b12202c Duration: 19.50 ms B
illed Duration: 100 ms Memory Size: 1024 MB Max Memory Used: 42 MB
START RequestId: 8a8114fd-49f1-11e6-8b91-f3f901377f77 Version: 7
2016-07-14 14:34:04.174 (-04:00) 8a8114fd-49f1-11e6-8b91-f3f901377f77 U
PDATED COUNT: 10114
END RequestId: 8a8114fd-49f1-11e6-8b91-f3f901377f77
REPORT RequestId: 8a8114fd-49f1-11e6-8b91-f3f901377f77 Duration: 22.45 ms B
illed Duration: 100 ms Memory Size: 1024 MB Max Memory Used: 42 MB
START RequestId: a08b79a0-49f1-11e6-ac13-e76221b4953c Version: 7
START RequestId: a0ef6c2b-49f1-11e6-baba-f3b821e99bfa Version: 7
^[[A^C
X ~/serverless-demo/prepared-demo/ping
j curl -L http://bit.ly/trek10-ping
{"count":10128}
~/serverless-demo/prepared-demo/ping
```

```
1. jshort@Jareds-MacBook-Pro: ~/serverless-demo/prepared-demo (zsh)
X ..rverless... 361 ~/serverless-demo/prepared-demo
X ..prepare... 362 ^ ls
X ~ (zsh) 363 _meta package.json s-project.json
node_modules ping s-resources-cf.json
~/serverless-demo/prepared-demo
^
```

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.

```
1. jshort@Jareds-MacBook-Pro: ~/serverless-demo/prepared-demo (zsh)
X ..rverless... 361 ~/serverless-demo/prepared-demo
X ..prepare... 362 ^ ls
X ~ (zsh) 363 _meta package.json s-project.json
node_modules ping s-resources-cf.json
~/serverless-demo/prepared-demo
^ npm install serverless-secrets --save
prepared-demo@0.0.1 /Users/jshort/serverless-demo/prepared-demo
├─ serverless-secrets@2.0.2
├─ aes-js@2.0.0
├─ aws-sdk@2.4.7
├─ jmespath@0.15.0
├─ sax@1.1.5
├─ xml2js@0.4.15
├─ xmlbuilder@2.6.2
├─ lodash@3.5.0
├─ bluebird@3.4.1
├─ lodash@4.13.1
├─ mpath@1.0.0
└─ ncp@2.0.0
~/serverless-demo/prepared-demo
^ nano s-project.json
```

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

```
1. nano s-project.json (nano)
X ..rverless... 361 GNU nano 2.0.6 File: s-project.json
X nano (na... 362 {
X ~ (zsh) 363   "name": "prepared-demo",
   "custom": {},
   "plugins": []
}
```

We are going to add in the code snippet below,

```
1. nano s-project.json (nano)
GNU nano 2.0.6      File: s-project.json      Modified
~ (zsh) %1
~ (zsh) %2
~ (zsh) %3
{
  "name": "prepared-demo",
  "custom": {
    "secrets": {
      "kms": {
        "default": "arn:aws:kms:us-east-1:738317252543:alias/credstash"
      }
    }
  },
  "plugins": [
    "serverless-secrets"
  ]
}
```

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.

```
1. sls --help (zsh)
~/serverless-demo/prepared-demo
~ (zsh) %1
~ (zsh) %2
~ (zsh) %3
$ sls --help
```

```
1. jshort@Jareds-MacBook-Pro: ~/serverless-demo/prepared-demo (zsh)
serverless.com, v0.5.5
Commands
* Serverless documentation: http://docs.serverless.com
* You can run commands with "serverless" or the shortcut "sls"
* Pass "--help" after any <context> <action> for contextual help
* Add "--debug" to any command for extra useful logs

project ..... create, init, install, remove
function ..... create, deploy, logs, remove, rollback, run
endpoint ..... deploy, remove
event ..... deploy, remove
dash ..... deploy, summary
stage ..... create, remove
region ..... create, remove
resources ..... deploy, diff, remove
plugin ..... create
variables ..... list, set, unset
secret ..... encrypt

~/serverless-demo/prepared-demo
$
```

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

```
1. jshort@Jareds-MacBook-Pro: ~/serverless-demo/prepared-demo (zsh)
X ..rverless... %1
X ..prepare... %2
X ~ (zsh) %3
project ..... create, init, install, remove
function ..... create, deploy, logs, remove, rollback, run
endpoint ..... deploy, remove
event ..... deploy, remove
dash ..... deploy, summary
stage ..... create, remove
region ..... create, remove
resources ..... deploy, diff, remove
plugin ..... create
variables ..... list, set, unset
secret ..... encrypt

~/serverless-demo/prepared-demo
$ sls secret --help

Actions for the 'secret' context:
Note: pass "--help" after any <context> <action> for contextual help

encrypt ..... Encrypt a secret from the CLI

~/serverless-demo/prepared-demo
$ sls secret encrypt --help
```

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

```
1. jshort@Jareds-MacBook-Pro: ~/serverless-demo/prepared-demo (zsh)
X ..rverless... %1
X ..prepare... %2
X ~ (zsh) %3
Encrypt a secret from the CLI

-p, --provider
    Provider to use for encryption (kms, kmsfile)

-t, --plaintext
    Plaintext string to encrypt

-f, --file
    Plaintext file to encrypt

-r, --region
    AWS Region (kms, kmsfile)

-a, --arn
    ARN or name from Custom config for the key to use ex: "prod", or "arn:aws:kms:us-east-1:123456789012:alias/MyKey"

-q, --quiet
    Don't display usage information when encrypting

~/serverless-demo/prepared-demo
$
```

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 encrypted.
Please copy paste the string below into your _meta folder, and include it
in the relevant s-function environment configurations.

kms::arn:aws:kms:us-east-1:738317252543:alias/credstash::CiDZ0VE8Sg9
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.

Serverless Framework Demo



Architectural Patterns

Automate Your AWS Environment



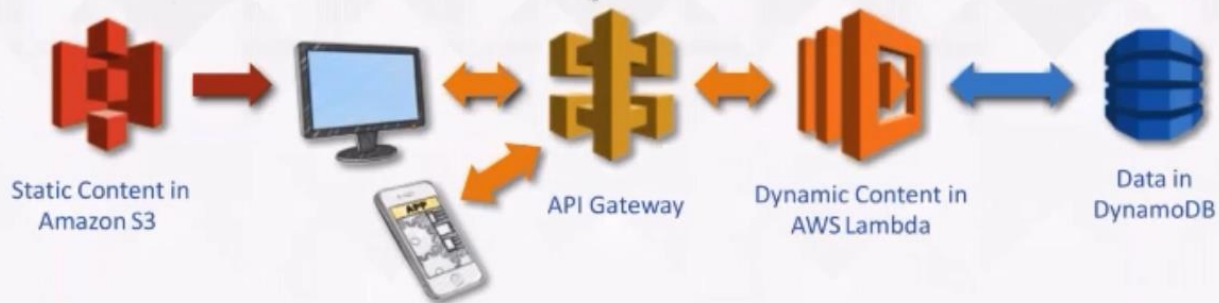
- ✓ ec2-run—instance
- ✓ ECS StartTask
- ✓ Beanstalk Update Application
- ✓ Kinesis SplitShard
- ✓ Any API Call

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.

Serverless Web & Mobile Apps

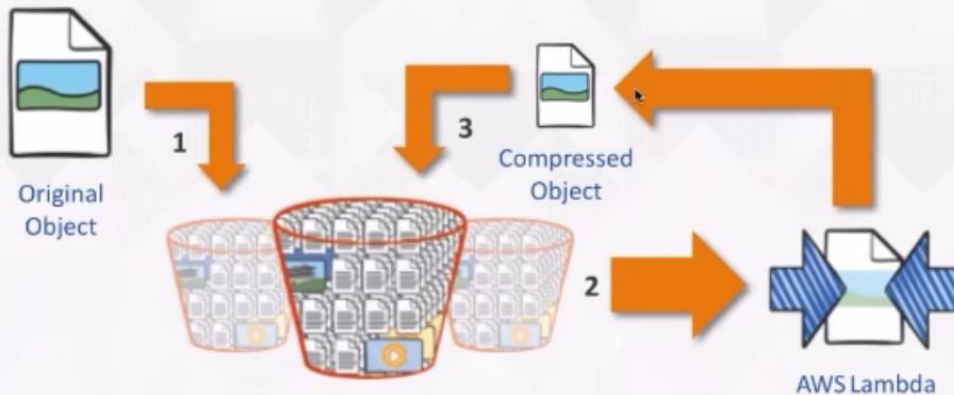
1. Fat client: Mobile app or single-page JS web app
2. JS and other static content in S3
3. APIs with API Gateway for http, Lambda for compute
4. Multiple options for auth
5. Amazon DynamoDB for NoSQL data storage

Take a look
at GraphQL



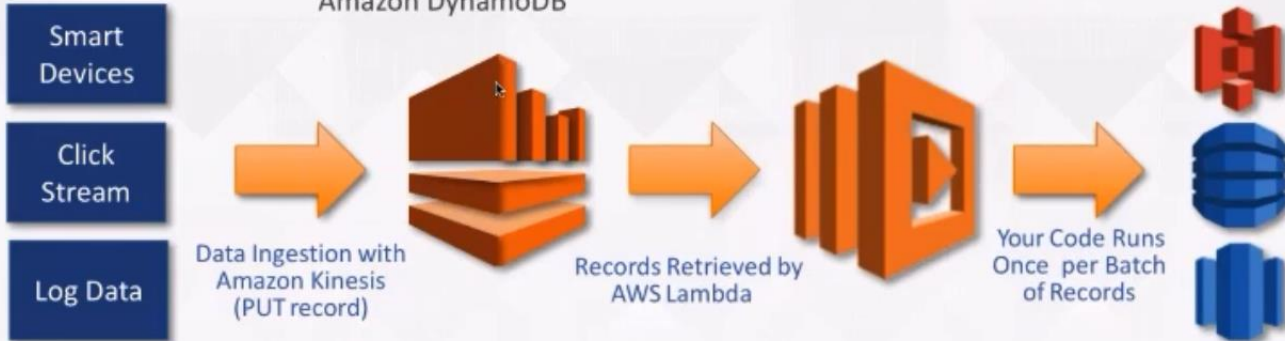
S3 Triggers: Media Handling, ETL

1. Pick the S3 Lambda blueprint
2. Select your bucket as the event source



Analytical Processing/ETL

- Real-time: Kinesis + Lambda consumer
- Near-real-time: S3 + Lambda trigger or Kinesis Firehose
- Store aggregated results in Amazon Redshift, Amazon S3, Amazon DynamoDB



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.

IoT Backend

1. Thing sends data over MQTT to AWS IOT Platform
2. Pub/sub model; any subscriber can be on the other end
3. IOT Rules/Actions to trigger other services: Lambda, Dynamo, SNS, Kinesis, etc.
4. IOT Device Shadow handles asynchronous state management



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.