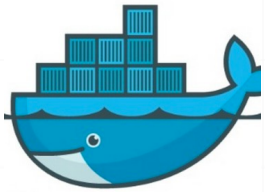

On-Prem, IaaS, etc.
Daenerys Stormborn
of House Targaryen
rightful heir to the Iron Throne, rightful queen of the Andals and First Men, Protector of the Seven Kingdoms, the Mother of Dragons, the Khaleesi of the Great Grass Sea, the Unburnt, the Breaker of Chains
Serverless
JON SNOW

Serverless Architectures with AWS Lambda and MongoDB Atlas

Sig Narváez
Sr. Solutions Architect
sig@mongodb.com
@SigNarvaez


Serverless

Where will my code run?



Big Iron

Commodity Hardware

Virtualized

Containers

Functions

Forbes: 5 factors fuelling Serverless Computing

1. Cloud services have matured
2. BaaS → *"SaaS-ification"*
3. API's are the glue
4. Containers – *now per function*
5. **SysOps → DevOps → NoOps**

Less Ops, More Engineering

https://www.forbes.com/sites/janakirammsv/2016/02/28/five-factors-that-are-fueling-serverless-computing-part-1

WE LIVE IN A SOFTWARE DEFINED ECONOMY

EVERY COMPANY IS A SOFTWARE COMPANY

DEVELOPERS ARE VALUABLE RESOURCE

BUILD VALUE – NOT OPERATIONS

Thoughtworks Technology Radar

Serverless architecture

TRIAL

Serverless architecture is an approach that replaces long-running virtual machines with ephemeral compute power that comes into existence on request and disappears immediately after use. Since the last Radar, we have had several teams put applications into production using a "serverless" style. Our teams like the approach, it's working well for them and we consider it a valid architectural choice. Note that serverless doesn't have to be an all-or-nothing approach: some of our teams have deployed a new chunk of their systems using serverless while sticking to a traditional architectural approach for other pieces.

Worth pursuing. It is important to understand how to build up this capability. Enterprises should try this technology on a project that can handle the risk.

Serverless architecture

NOV 2016

TRIAL

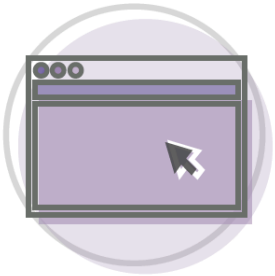
Serverless architecture is an approach that replaces long-running virtual machines with ephemeral compute power that comes into existence on request and disappears immediately after use. Since the last Radar, we have had several teams put applications into production using a "serverless" style. Our teams like the approach, it's working well for them and we consider it a valid architectural choice. Note that serverless doesn't have to be an all-or-nothing approach: some of our teams have deployed a new chunk of their systems using serverless while sticking to a traditional architectural approach for other pieces.

APR 2016

ASSESS

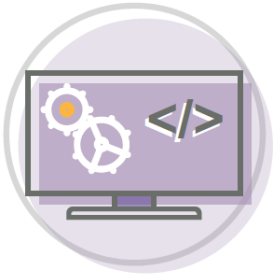
Serverless architecture replaces long-running virtual machines with ephemeral compute power that comes into existence on request and disappears immediately after use. Examples include Firebase and AWS Lambda. Use of this architecture can mitigate some security concerns such as security patching and SSH access control, and can make much more efficient use of compute resources. These systems cost very little to operate and

Common use cases



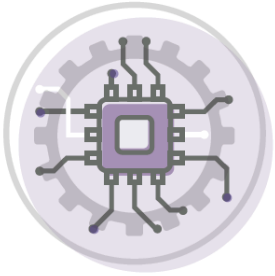
Web applications

- Static websites
- Dynamic web apps
- Packages for Flask and Express



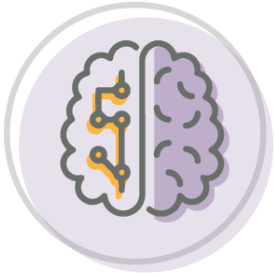
Back ends

- Apps & services
- Mobile
- IoT



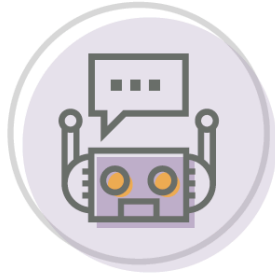
Data processing

- Real time
- MapReduce
- Batch



Amazon Alexa

- Powering voice-enabled apps
- Alexa Skills Kit

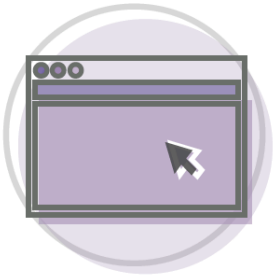


Chatbots

- Powering chatbot logic

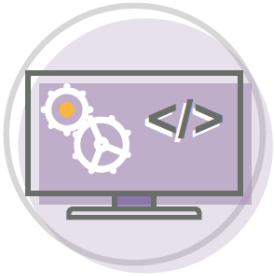
AWS re:Invent 2016: The State of Serverless Computing (SVR311)

Common use cases



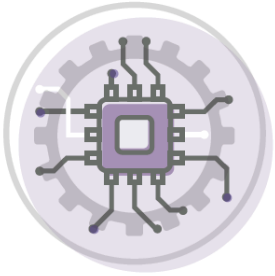
Web applications

- Static websites
- Dynamic web apps
- Packages for Flask and Express



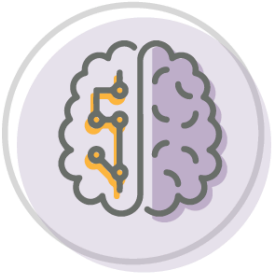
Back ends

- Apps & services
- Mobile
- IoT



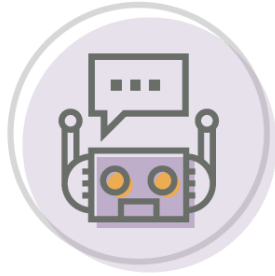
Data processing

- Real time
- MapReduce
- Batch



Amazon Alexa

- Powering voice-enabled apps
- Alexa Skills Kit

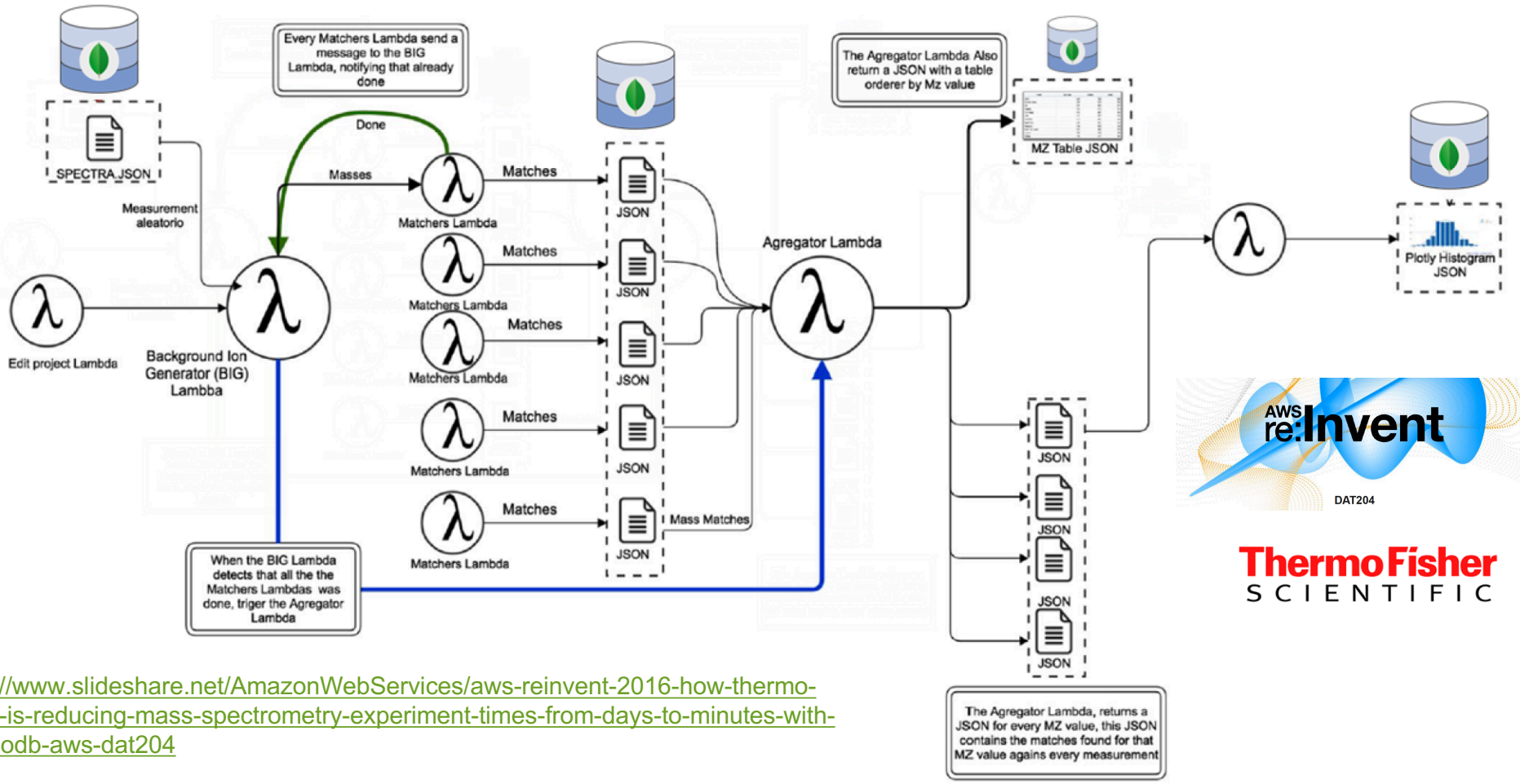


Chatbots

- Powering chatbot logic

AWS re:Invent 2016: The State of Serverless Computing (SVR311)

Parallel data processing



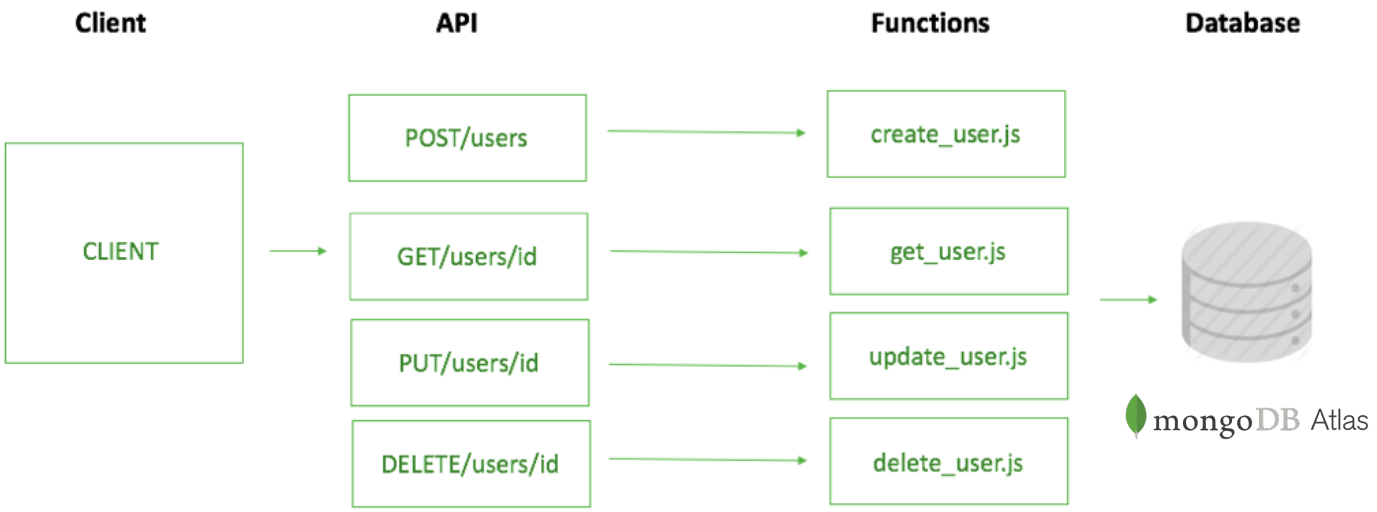
AWS re:Invent DAT204

ThermoFisher SCIENTIFIC

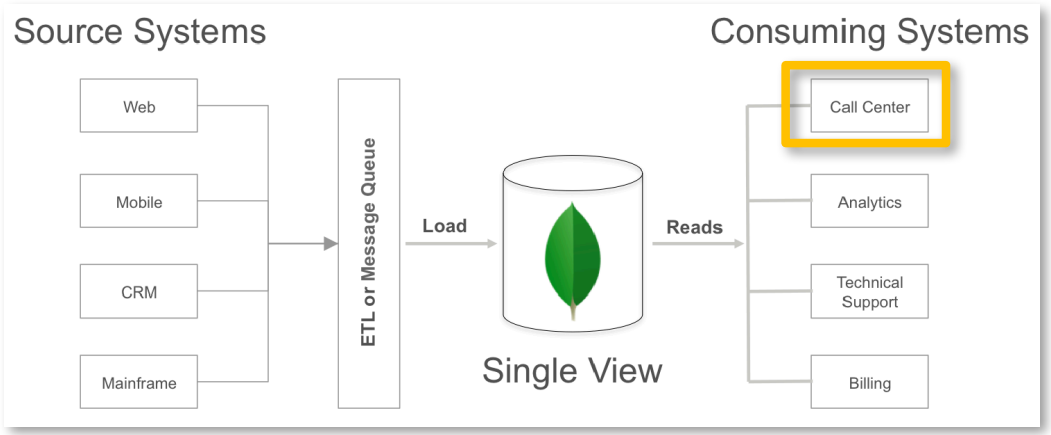
https://www.slideshare.net/AmazonWebServices/aws-reinvent-2016-how-thermo-fisher-is-reducing-mass-spectrometry-experiment-times-from-days-to-minutes-with-mongodb-aws-dat204

Microservices

Serverless Microservice



Customer Single View - Insurance Industry (hypothetical)



High-level architecture of a single view platform

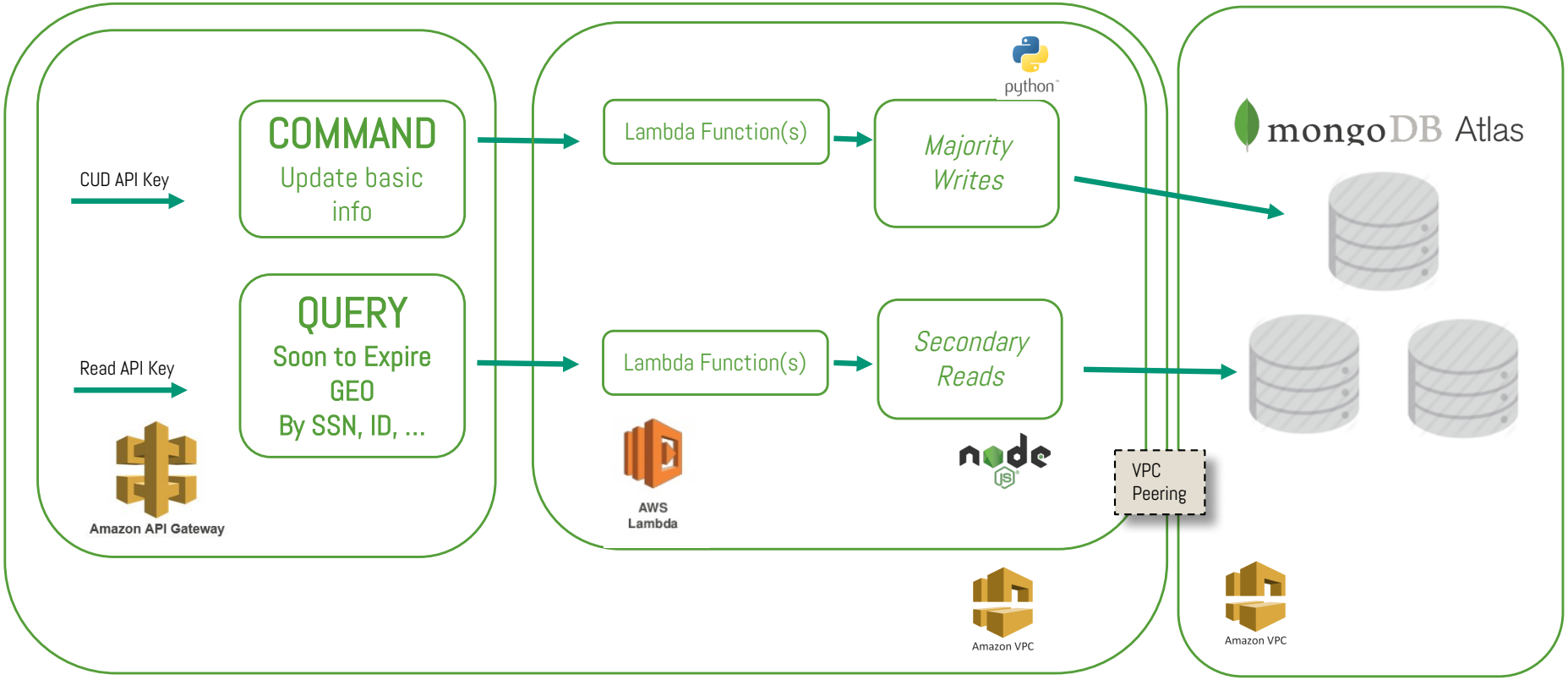
Shape

- Person
- Insurance Policies
  - Shape changes per policy type
- Addresses

Operations via API

- **GET** Customers with soon-to-expire policies, within a geo radius
- **GET** Customers / by SSN, id, etc.
- PATCH Update basic contact info (cell, email, ...)

CQRS on Serverless Microservices



MongoDB Atlas & AWS

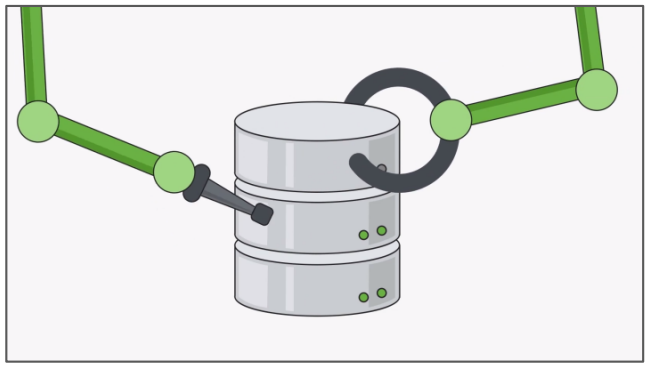
Build it!

MongoDB Atlas

amazon web services

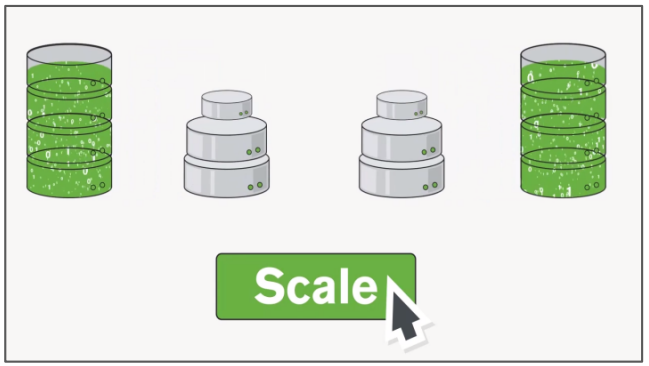
Google Cloud Platform

Microsoft Azure

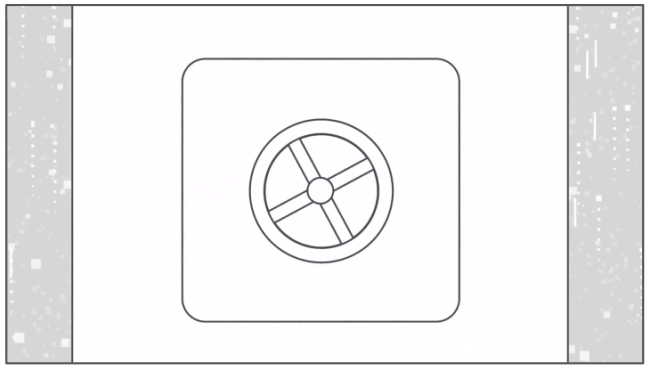


Automated Service

On-demand DBaaS

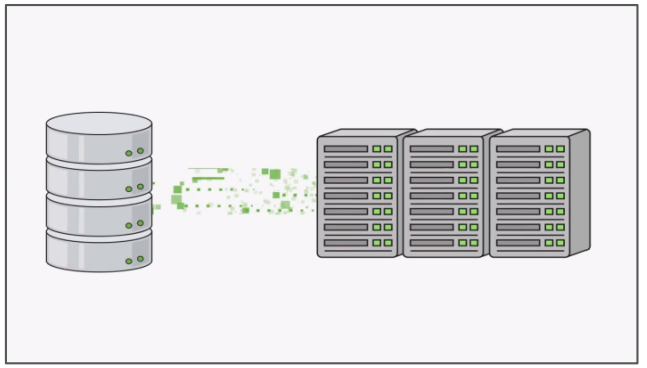


Elastic Scalability



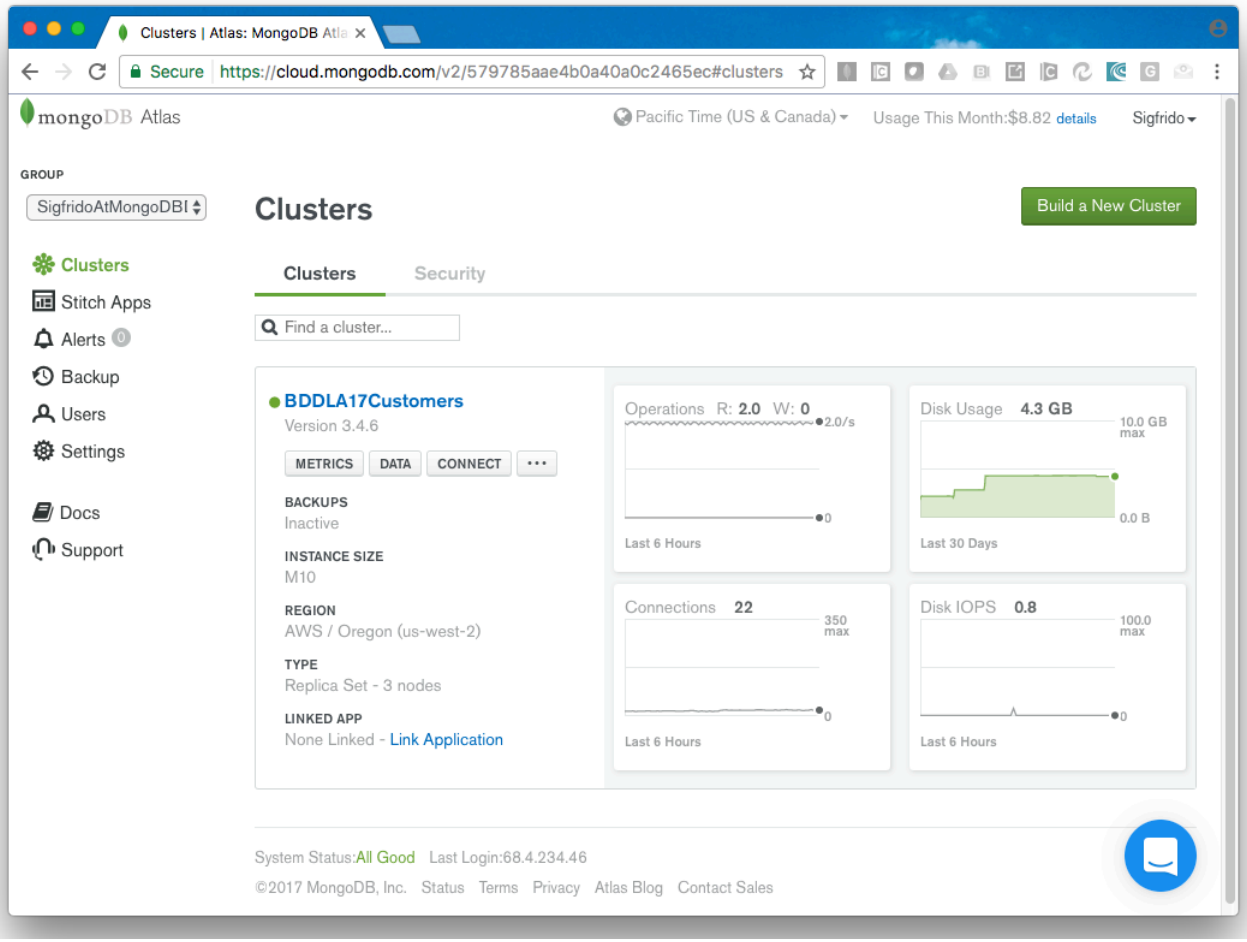
Secure

Highly Available



Disaster Recovery

Setup MongoDB Atlas - M10+ *Need assigned AWS region*



Generate dataset (optional)

mgeneratejs

- https://github.com/rueckstiess/mgeneratejs
- npm install -g mgeneratejs
- Create template – generate data
- Upload to Atlas via mongoimport
  - Hint: get connection string from Atlas UI!
- Browse with Compass

```
mgeneratejs -n 100 CustomerSingleView.json | mongoimport --host "YOUR ATLAS CLUSTER" --numInsertionWorkers 4
--db SingleView--collection Customers --authenticationDatabase admin --ssl --username YOURUSER --password YOURPASSWORD
```

Template (CustomerSingleView.json)

```
{
  "firstname": "$first",
  "lastname": "$last",
  "cell": { "$number": { "min": 1111111111, "max": 9999999999 }},
  "email": "$email",
  "yob": { "$date": { "min": "1930-01-01", "max": "2016-12-31" }},
  "gender": "$gender",
  "address": {
    "number": { "$number": { "min": 1, "max": 9999 }},
    "street": { "$street": { "country": "us" }},
    "city": "$city",
    "state": { "$state": { "country": "us" }},
    "zip": "$zip",
    "location": { "$point": { "long_lim": [-118.668469, -82.062023], "lat_lim"
  },
  "policies": { "$array": { "of": { "$choose": { "from": [
            {
```

mongoDB

Required AWS Services

IAM

- Role with Lambda execute policies

VPC

- VPC Peering Connection
- Security Group

Lambda

- Set VPC, Security Group and IAM role
- Upload deployment package (.zip)

API Gateway

- API definition (Resources & HTTP Methods)
- Map Routes to Lambda functions
- API Keys & Usage Plans

VPC Peering

VPC Peering

USERS | IP WHITELIST | PEERING

+ NEW PEERING CONNECTION

ATLAS CIDR BLOCK: **192.168.248.0/21**

VPC ID	Status	Peering ID	VPC CIDR	
vpc-854200e3	Available	pcx-33c3595a	172.31.0.0/16	TERMINATE

Atlas

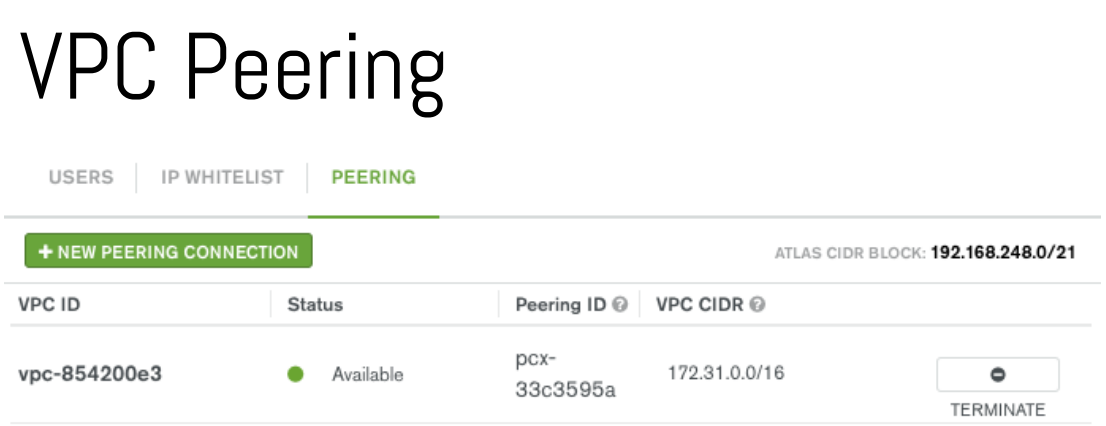
USERS | IP WHITELIST | PEERING

+ ADD IP ADDRESS

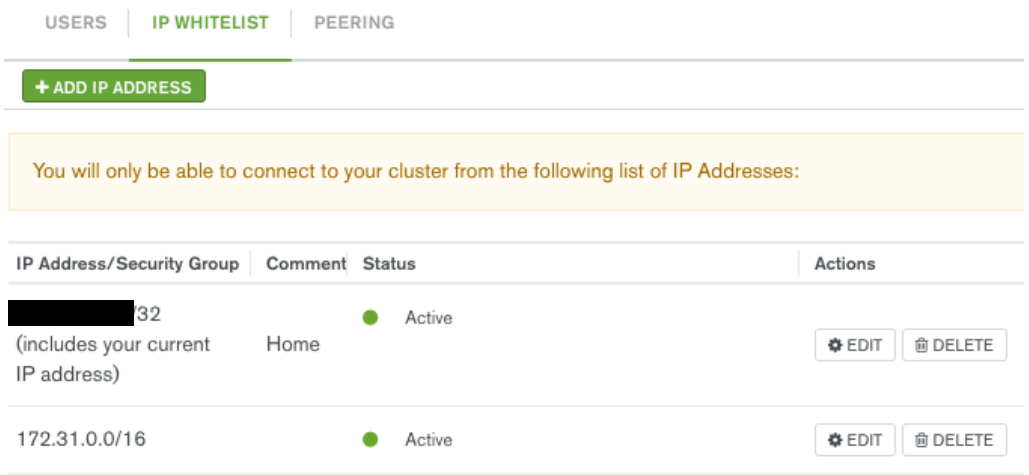
You will only be able to connect to your cluster from the following list of IP Addresses:

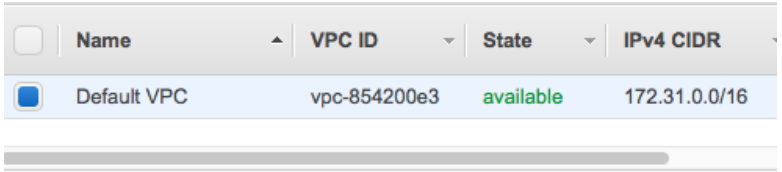
IP Address/Security Group	Comment	Status	Actions
[redacted]32 (includes your current IP address)	Home	Active	EDIT DELETE
172.31.0.0/16		Active	EDIT DELETE

VPC Peering

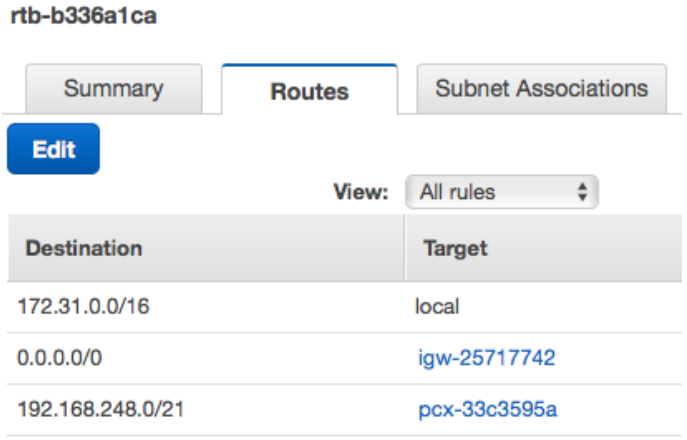


Atlas

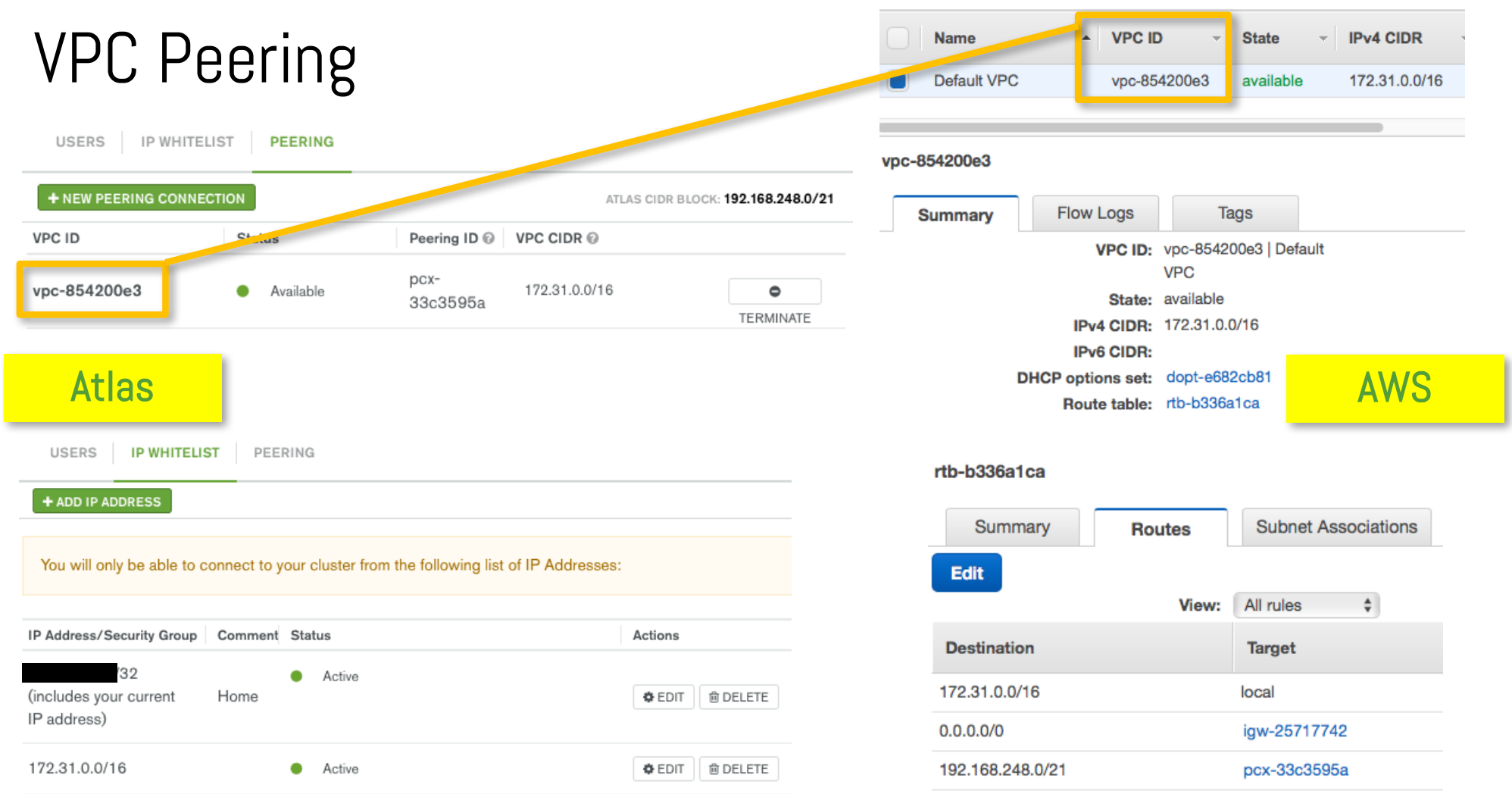




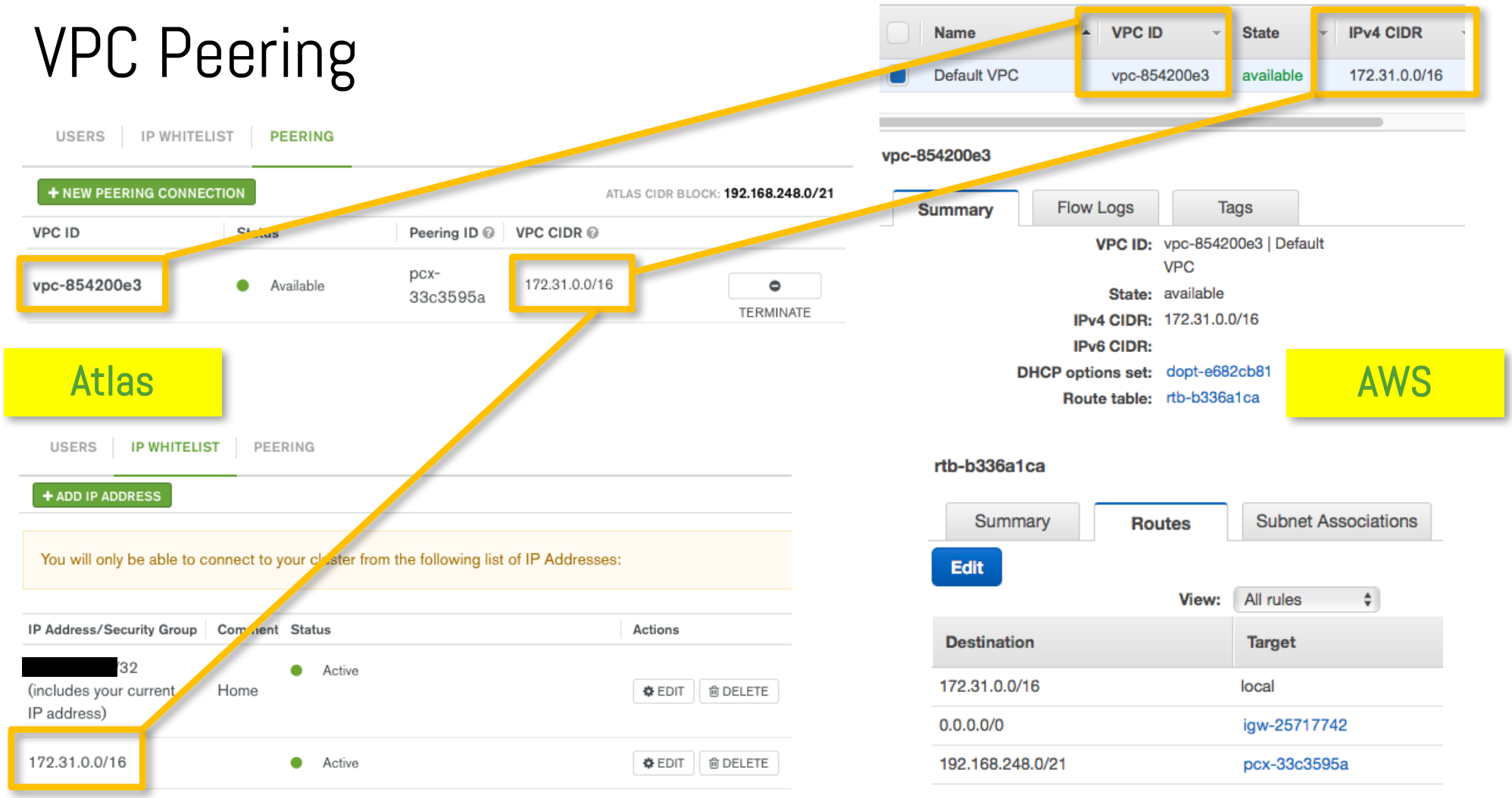
AWS



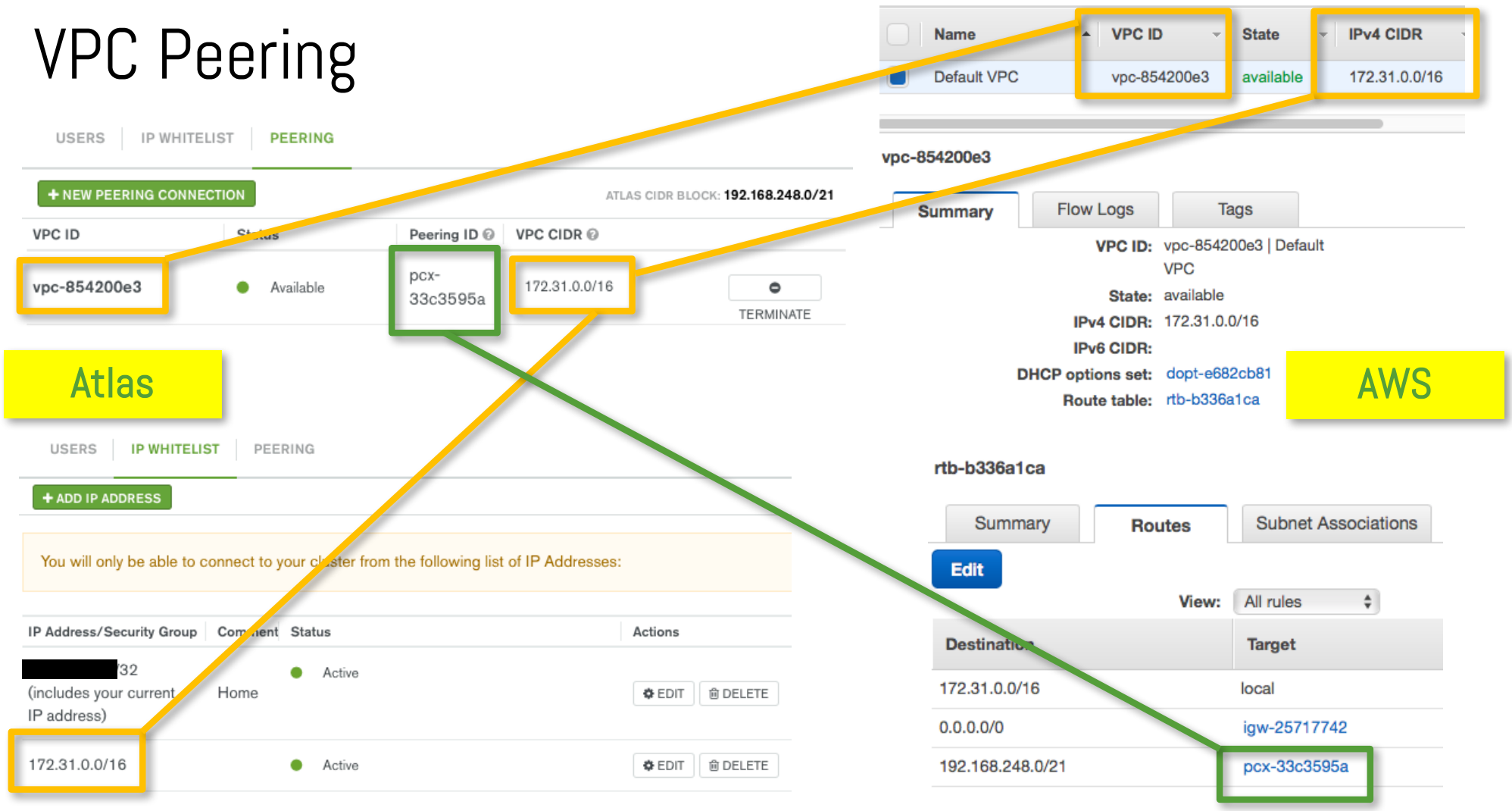
VPC Peering


USERS
IP WHITELIST
PEERING
+ NEW PEERING CONNECTION
ATLAS CIDR BLOCK: 192.168.248.0/21
VPC ID
Peering ID
VPC CIDR
vpc-854200e3
Available
pcx-33c3595a
172.31.0.0/16
TERMINATE
Atlas
USERS
IP WHITELIST
PEERING
+ ADD IP ADDRESS
You will only be able to connect to your cluster from the following list of IP Addresses:
IP Address/Security Group
Comment
Status
Actions
[redacted]/32
(includes your current IP address)
Home
Active
EDIT
DELETE
172.31.0.0/16
Active
EDIT
DELETE
Name
VPC ID
State
IPv4 CIDR
Default VPC
vpc-854200e3
available
172.31.0.0/16
vpc-854200e3
Summary
Flow Logs
Tags
VPC ID: vpc-854200e3 | Default VPC
State: available
IPv4 CIDR: 172.31.0.0/16
IPv6 CIDR:
DHCP options set: dopt-e682cb81
Route table: rtb-b336a1ca
AWS
rtb-b336a1ca
Summary
Routes
Subnet Associations
Edit
View: All rules
Destination
Target
172.31.0.0/16
local
0.0.0.0/0
igw-25717742
192.168.248.0/21
pcx-33c3595a

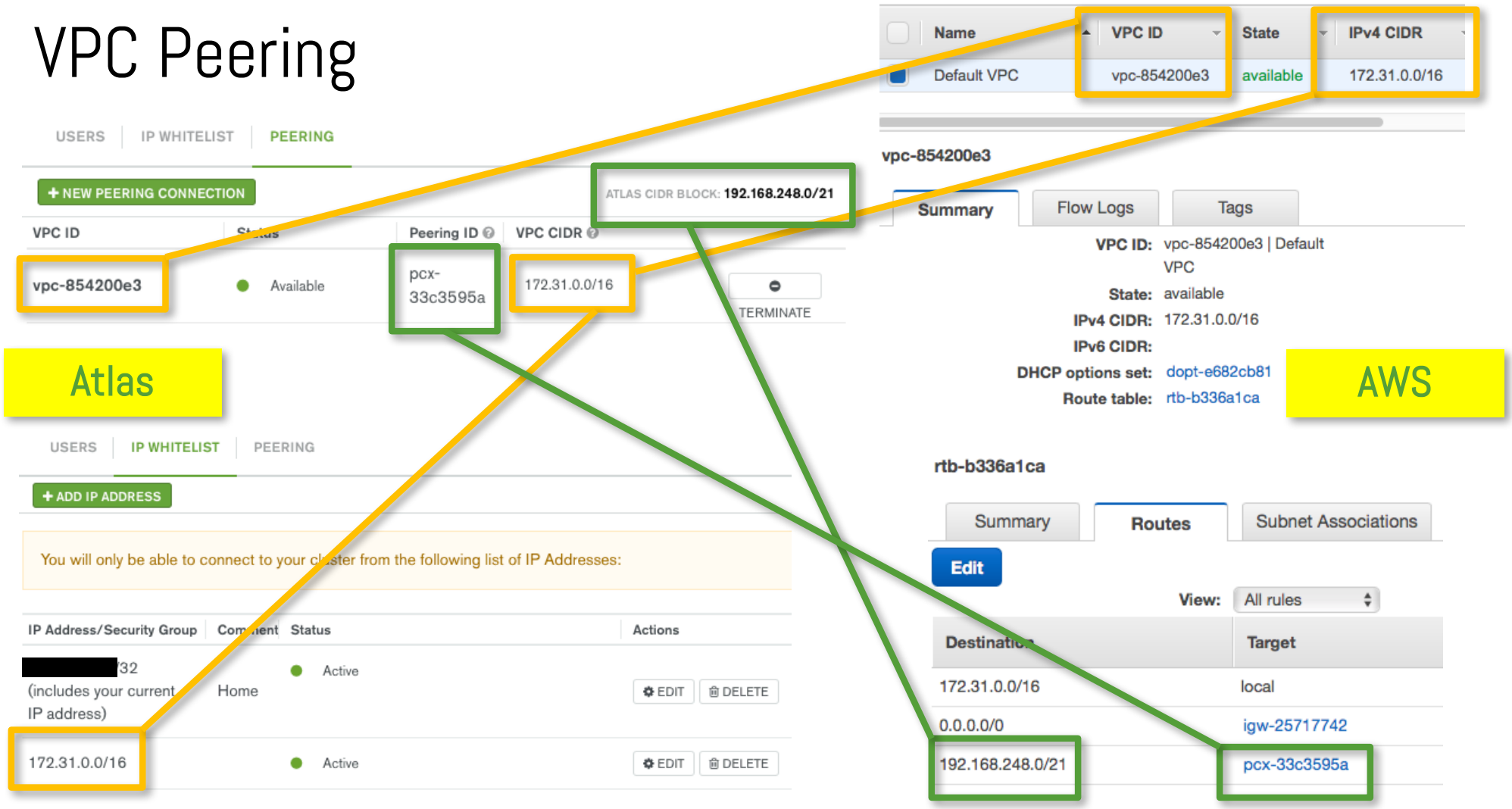
VPC Peering


USERS
IP WHITELIST
PEERING
+ NEW PEERING CONNECTION
ATLAS CIDR BLOCK: 192.168.248.0/21
VPC ID
Peering ID
VPC CIDR
vpc-854200e3
Available
pcx-33c3595a
172.31.0.0/16
TERMINATE
Atlas
USERS
IP WHITELIST
PEERING
+ ADD IP ADDRESS
You will only be able to connect to your cluster from the following list of IP Addresses:
IP Address/Security Group
Comment
Status
Actions
[redacted]/32 (includes your current IP address)
Home
Active
EDIT
DELETE
172.31.0.0/16
Active
EDIT
DELETE
Name
VPC ID
State
IPv4 CIDR
Default VPC
vpc-854200e3
available
172.31.0.0/16
vpc-854200e3
Summary
Flow Logs
Tags
VPC ID: vpc-854200e3 | Default VPC
State: available
IPv4 CIDR: 172.31.0.0/16
IPv6 CIDR:
DHCP options set: dopt-e682cb81
Route table: rtb-b336a1ca
AWS
rtb-b336a1ca
Summary
Routes
Subnet Associations
Edit
View: All rules
Destination
Target
172.31.0.0/16
local
0.0.0.0/0
igw-25717742
192.168.248.0/21
pcx-33c3595a

VPC Peering


USERS
IP WHITELIST
PEERING
+ NEW PEERING CONNECTION
ATLAS CIDR BLOCK: 192.168.248.0/21
VPC ID
Peering ID
VPC CIDR
vpc-854200e3
Available
pcx-33c3595a
172.31.0.0/16
TERMINATE
Atlas
USERS
IP WHITELIST
PEERING
+ ADD IP ADDRESS
You will only be able to connect to your cluster from the following list of IP Addresses:
IP Address/Security Group
Comment
Status
Actions
[redacted]/32
(includes your current IP address)
Home
Active
EDIT
DELETE
172.31.0.0/16
Active
EDIT
DELETE
Name
VPC ID
State
IPv4 CIDR
Default VPC
vpc-854200e3
available
172.31.0.0/16
vpc-854200e3
Summary
Flow Logs
Tags
VPC ID: vpc-854200e3 | Default VPC
State: available
IPv4 CIDR: 172.31.0.0/16
IPv6 CIDR:
DHCP options set: dopt-e682cb81
Route table: rtb-b336a1ca
AWS
rtb-b336a1ca
Summary
Routes
Subnet Associations
Edit
View: All rules
Destination
Target
172.31.0.0/16
local
0.0.0.0/0
igw-25717742
192.168.248.0/21
pcx-33c3595a

VPC Peering


USERS
IP WHITELIST
PEERING
+ NEW PEERING CONNECTION
ATLAS CIDR BLOCK: 192.168.248.0/21
VPC ID
Peering ID
VPC CIDR
vpc-854200e3
Available
pcx-33c3595a
172.31.0.0/16
TERMINATE
Atlas
USERS
IP WHITELIST
PEERING
+ ADD IP ADDRESS
You will only be able to connect to your cluster from the following list of IP Addresses:
IP Address/Security Group
Comment
Status
Actions
[redacted]/32
(includes your current IP address)
Home
Active
EDIT
DELETE
172.31.0.0/16
Active
EDIT
DELETE
Name
VPC ID
State
IPv4 CIDR
Default VPC
vpc-854200e3
available
172.31.0.0/16
vpc-854200e3
Summary
Flow Logs
Tags
VPC ID: vpc-854200e3 | Default VPC
State: available
IPv4 CIDR: 172.31.0.0/16
IPv6 CIDR:
DHCP options set: dopt-e682cb81
Route table: rtb-b336a1ca
AWS
rtb-b336a1ca
Summary
Routes
Subnet Associations
Edit
View: All rules
Destination
Target
172.31.0.0/16
local
0.0.0.0/0
igw-25717742
192.168.248.0/21
pcx-33c3595a

Lambda

Connections & Containers

… AWS Lambda maintains the container for some time in anticipation of another Lambda function invocation. … the service freezes the container after a function completes, and thaws the container for reuse. If AWS Lambda chooses to reuse the container, this has the following implications:

- Any declarations in your Lambda function code (outside the handler code, see Programming Model) remains initialized, providing additional optimization when the function is invoked again. For example, if your Lambda function establishes a database connection, instead of reestablishing the connection, the original connection is used in subsequent invocations. You can add logic in your code to check if a connection already exists before creating one.

http://docs.aws.amazon.com/lambda/latest/dg/lambda-introduction.html

Python lambda function skeleton

```
from __future__ import print_function
import json
import pymongo

# Atlas connection outside of Lambda handler(s)
print('=== CONNECTING TO MONGODB ATLAS ===') # logs to CloudWatch
atlasUri = os.environ['MONGODB_ATLAS_CLUSTER_URI']

MongoClient = pymongo.MongoClient(atlas_uri)
print("=== DONE - DB NAME: " + MongoClient.get_default_database().name + " ===")

# ================================================================
def lambda_handler(event, context):
# ================================================================

    # get handle to collection
    coll = MongoClient.MyDB.MyColl
    ...
```

Node.js lambda function skeleton

```
'use strict';
var MongoClient = require('mongodb').MongoClient;
var assert = require('assert');
var ObjectId = require('mongodb').ObjectID;

// Atlas connection outside of Lambda handler
const atlas_uri = ">>> Atlas conn string here <<<"; // to-do read from AWS keys

let cachedDb = null;

exports.handler = (event, context, callback) => {
    // Set to false to allow re-use of database connections across lambda calls
    // see http://docs.aws.amazon.com/lambda/latest/dg/nodejs-prog-model-context.html
    context.callbackWaitsForEmptyEventLoop = false;

    try {
        if (cachedDb && cachedDb.serverConfig.isConnected()) {
            //Execute Query
            executeQuery(cachedDb, {}, callback);
        }
        else {
            console.log("=== CONNECTING TO MONGODB ATLAS ==="); // Log to CloudWatch
            MongoClient.connect(atlas_uri, function (err, db) {
                if (err) {
                    console.log(`the error is ${err}.`, err);
                    process.exit(1);
                }
                cachedDb = db;
```

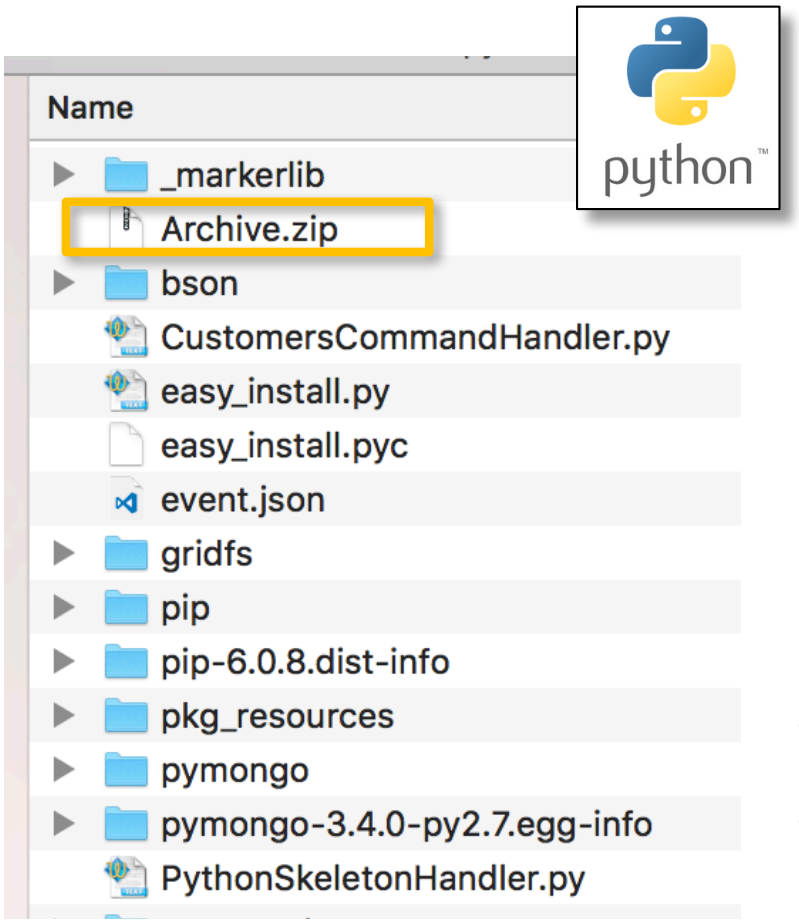
Local Emulators : Test on EC2 instance against Atlas

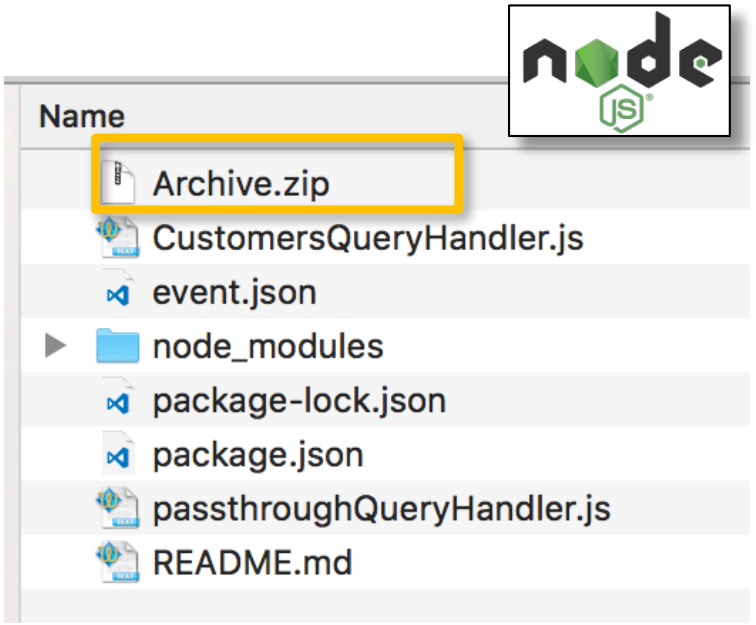
python-lambda-local at https://pypi.python.org/pypi/python-lambda-local

lambda-local (node.js) at https://www.npmjs.com/package/lambda-local

mongoDB

Code packaging





http://docs.aws.amazon.com/lambda/latest/dg/deployment-package-v2.html

Lambda functions

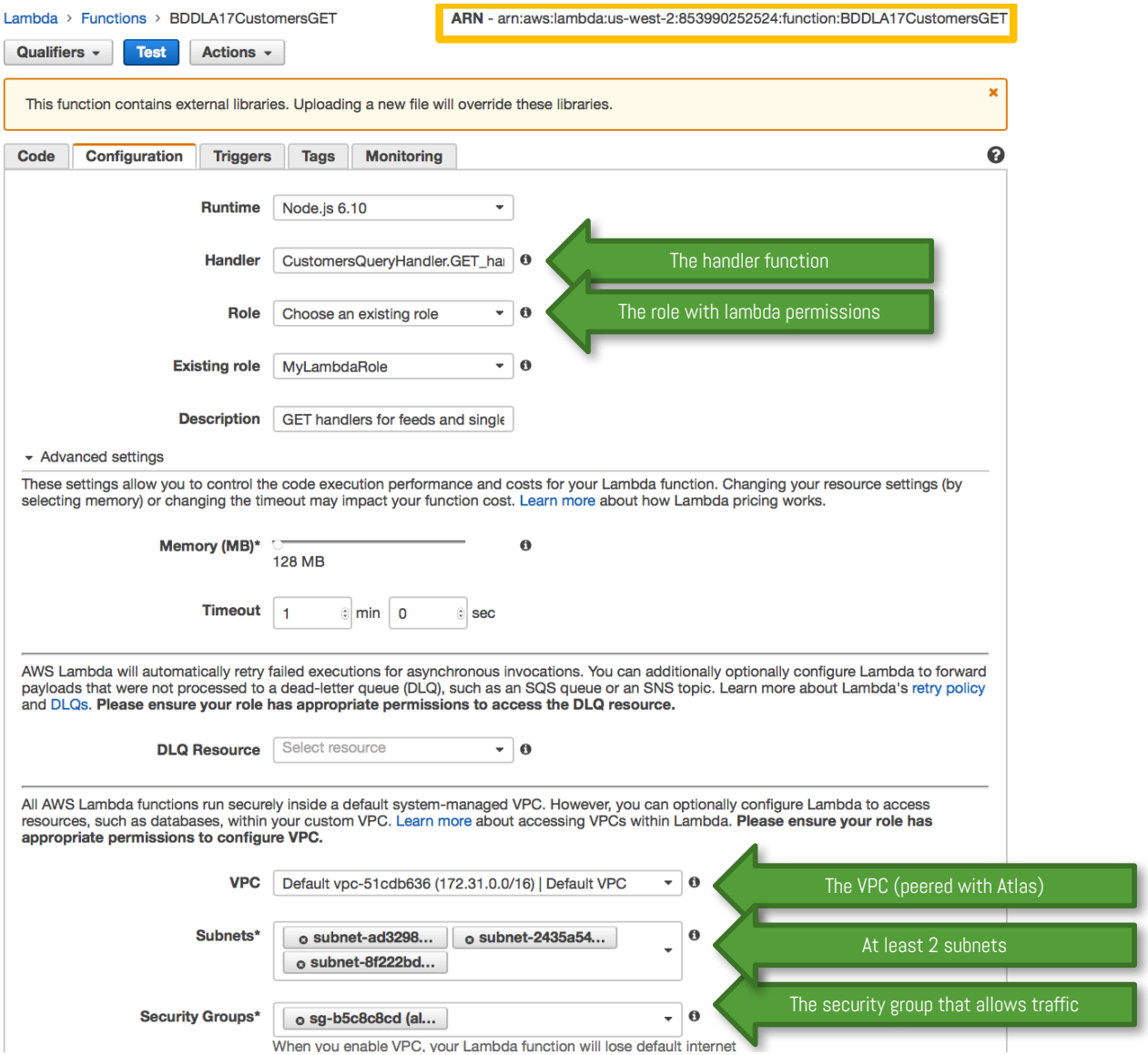
Lambda > Functions

Create a Lambda function | Actions

Filter by tags and attributes or search by keyword

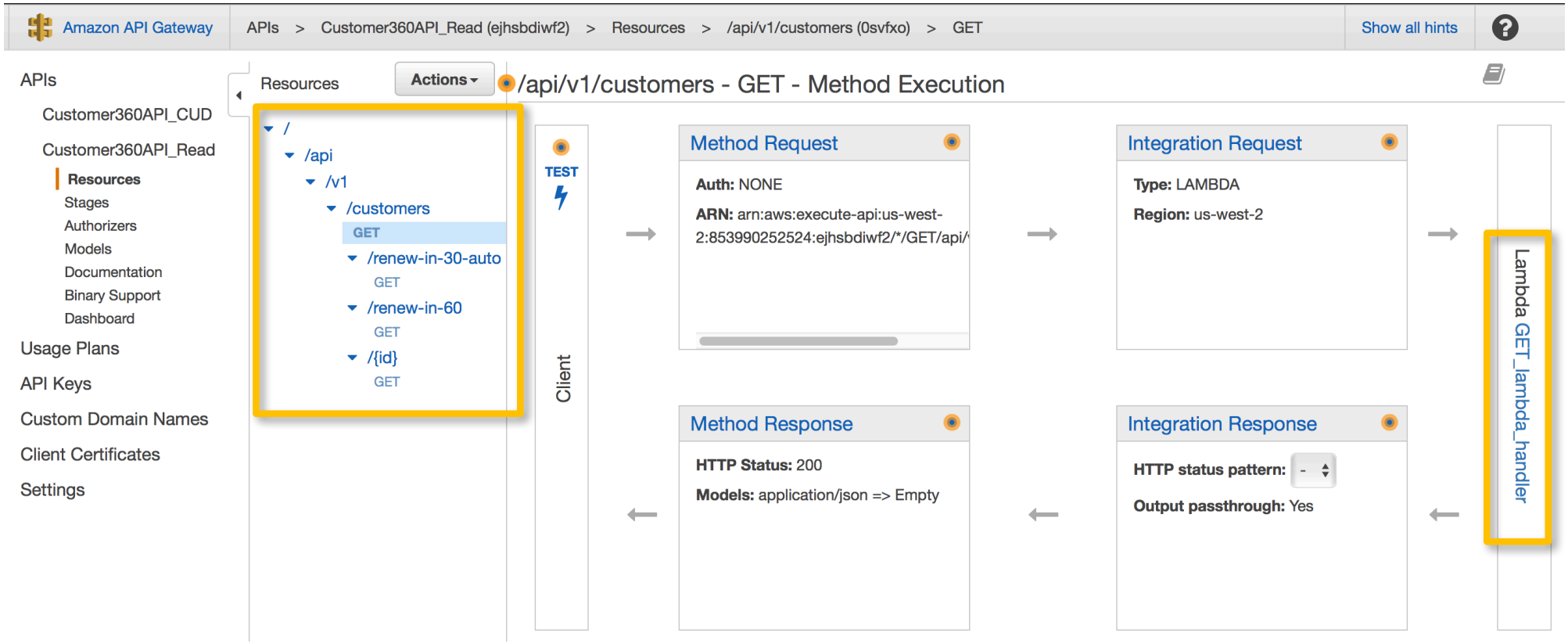
Function name	Description	Runtime	Code size
BDDLA17CustomersPATCH	PATCH handlers for update basic info	Python 2.7	3.8 MB
BDDLA17CustomersGET	GET handlers for feeds and single queries	Node.js 6.10	620.3 kB
BDDLA17CustomersRenewalsGET	GET handlers for GEO & Date queries	Node.js 6.10	620.3 kB

Upload & configure function

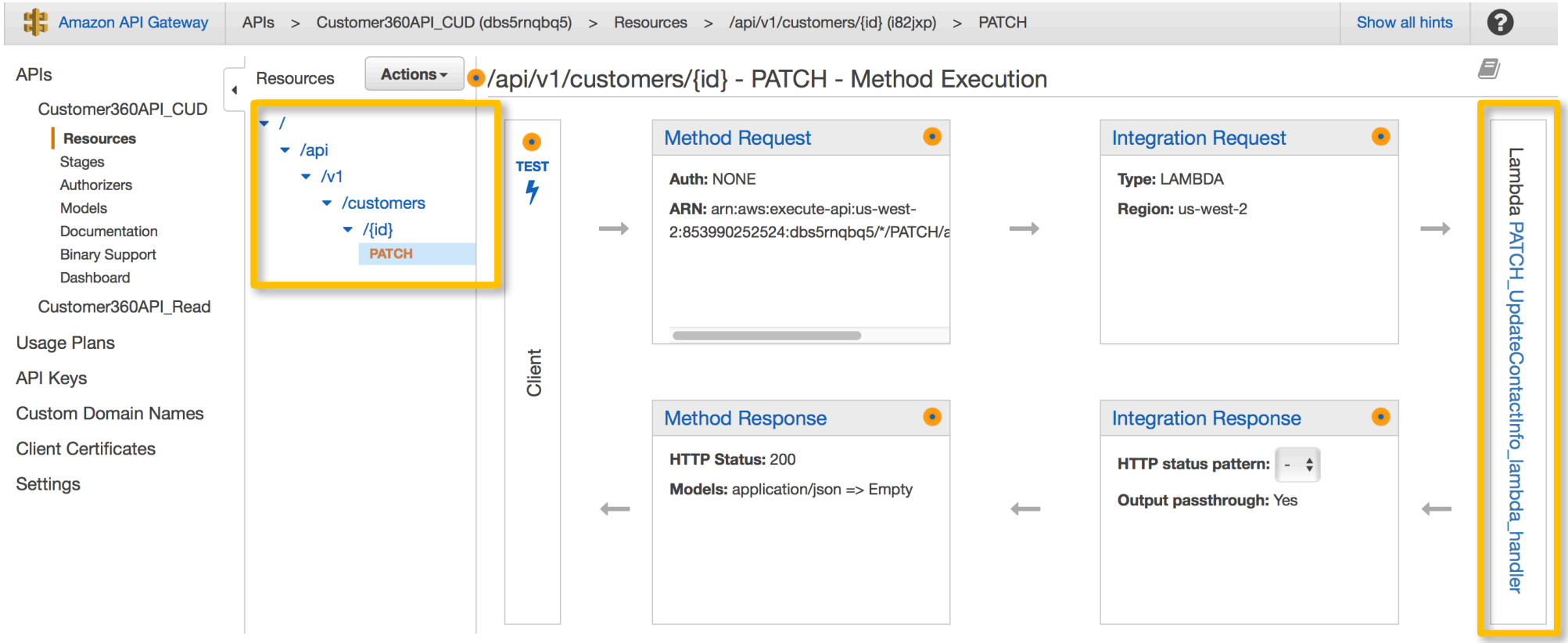


API Gateway

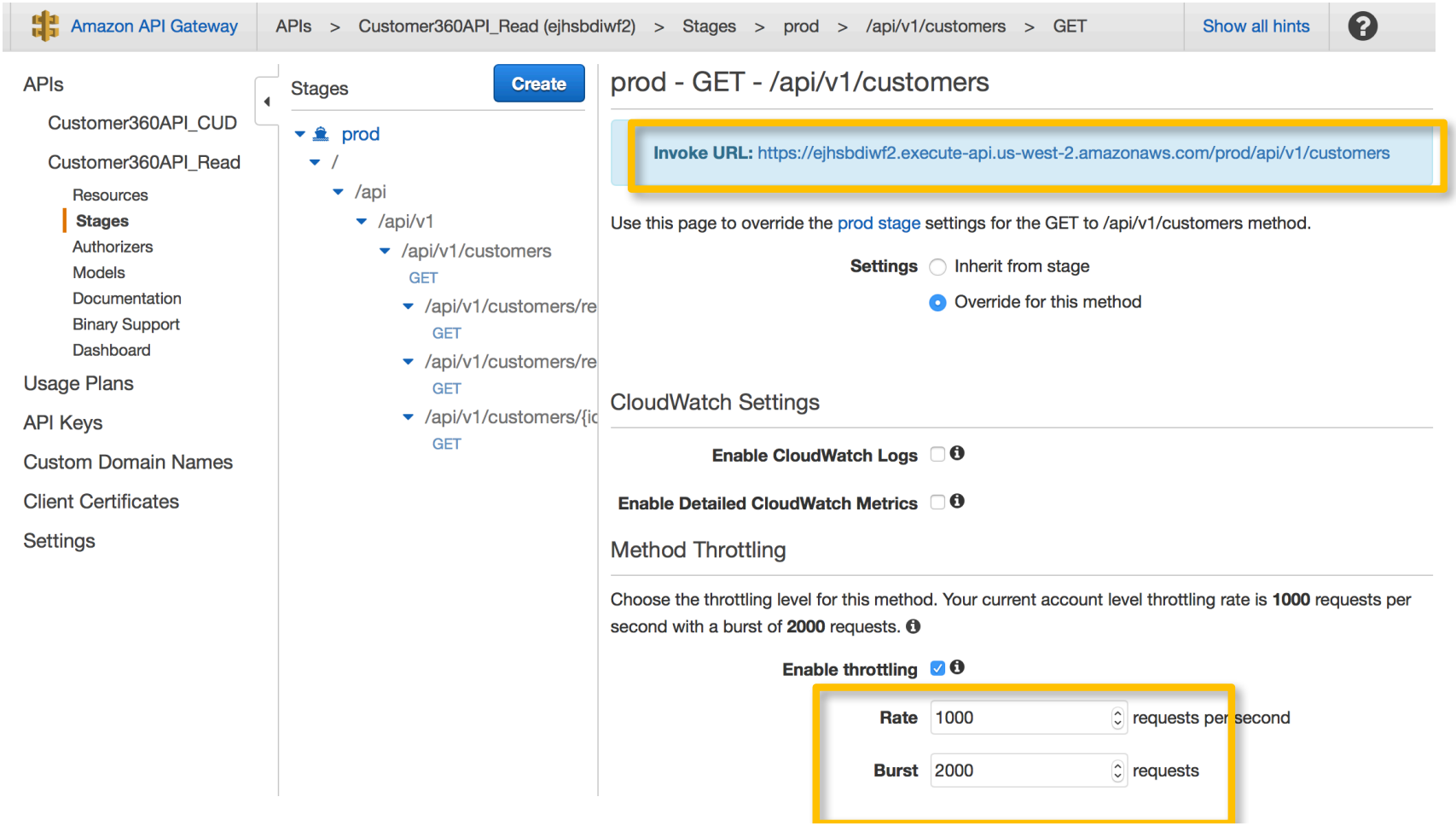
Read API – GET /api/v1/customers



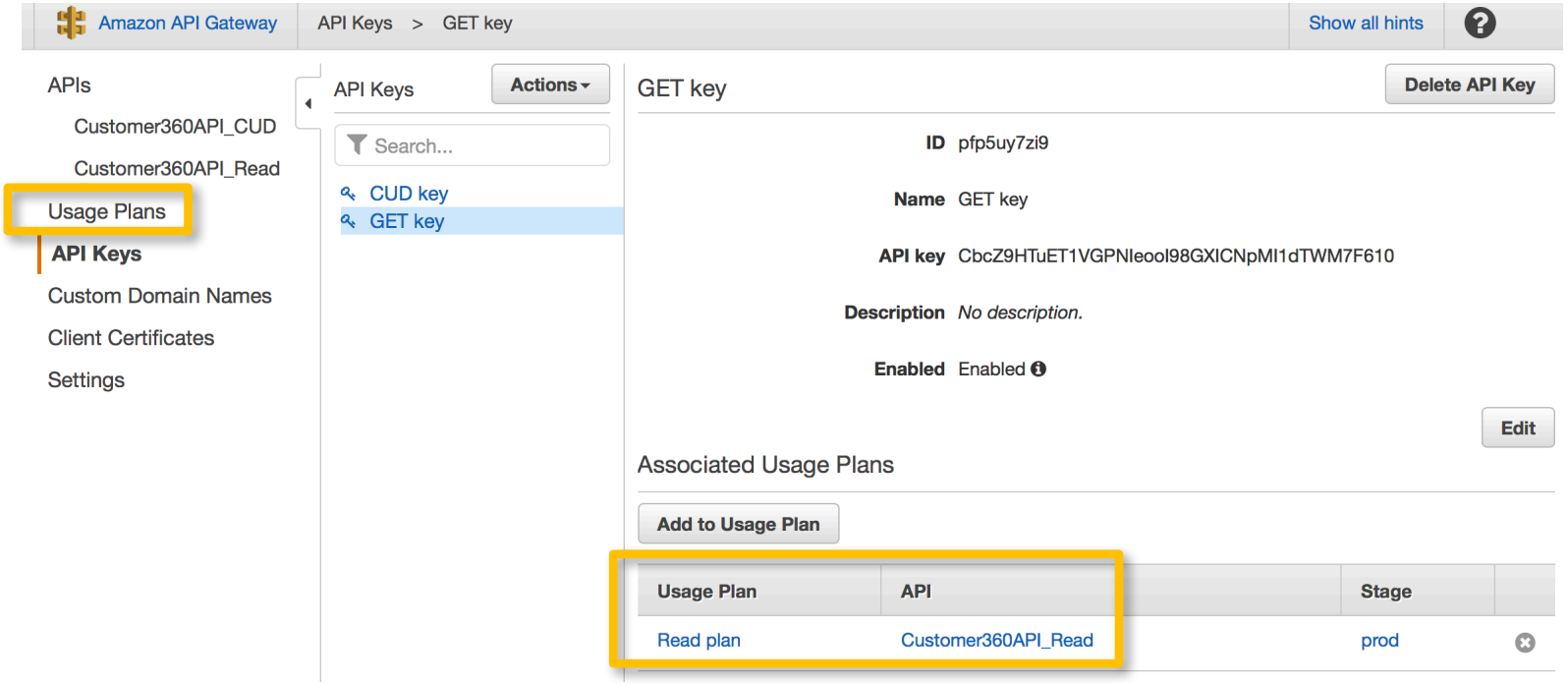
CUD API - PATCH /api/v1/customers



Deploying the API

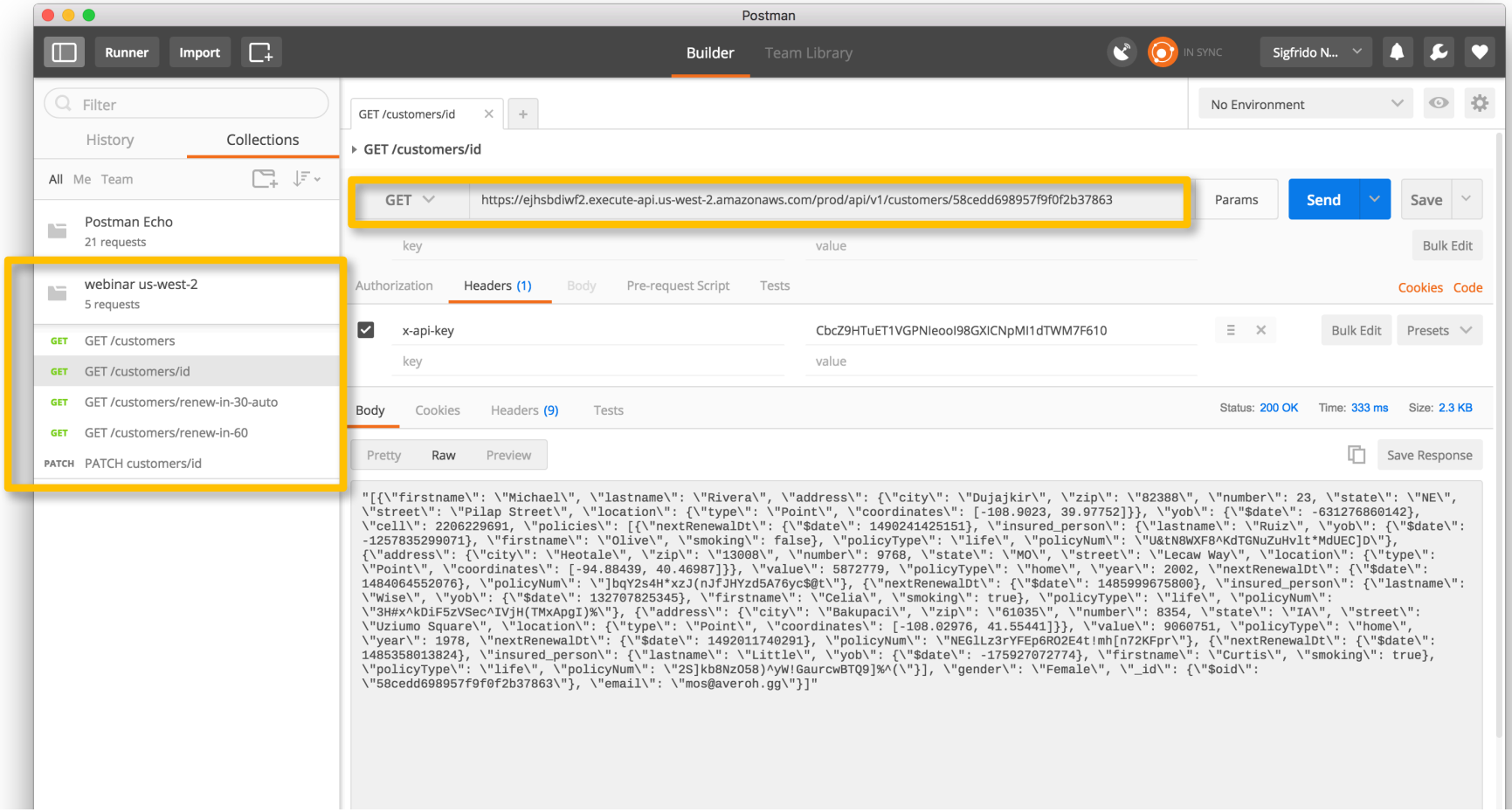


Access and throttling via API Keys

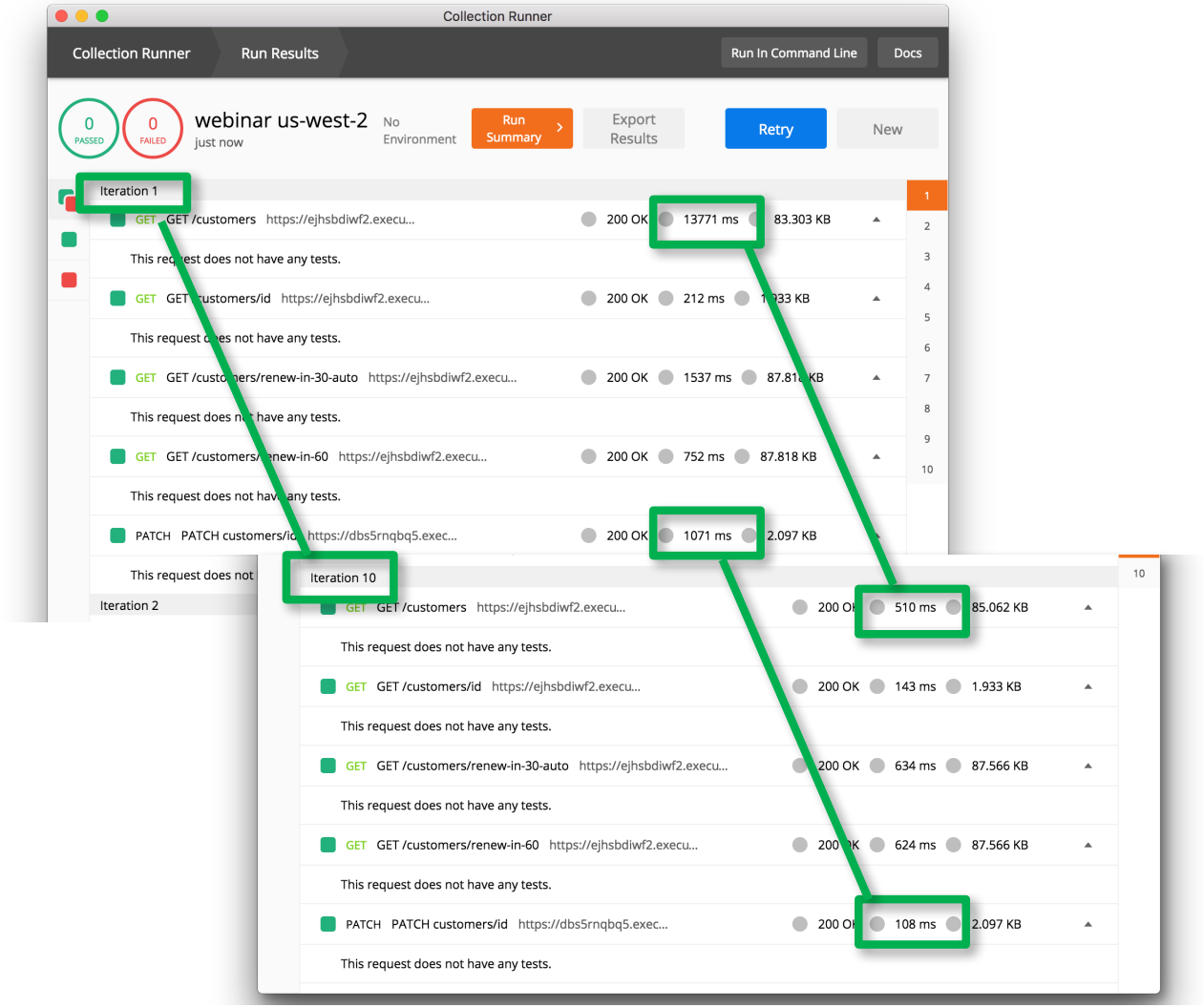


Test!

Test with Postman



Load test too!



AWS CloudWatch

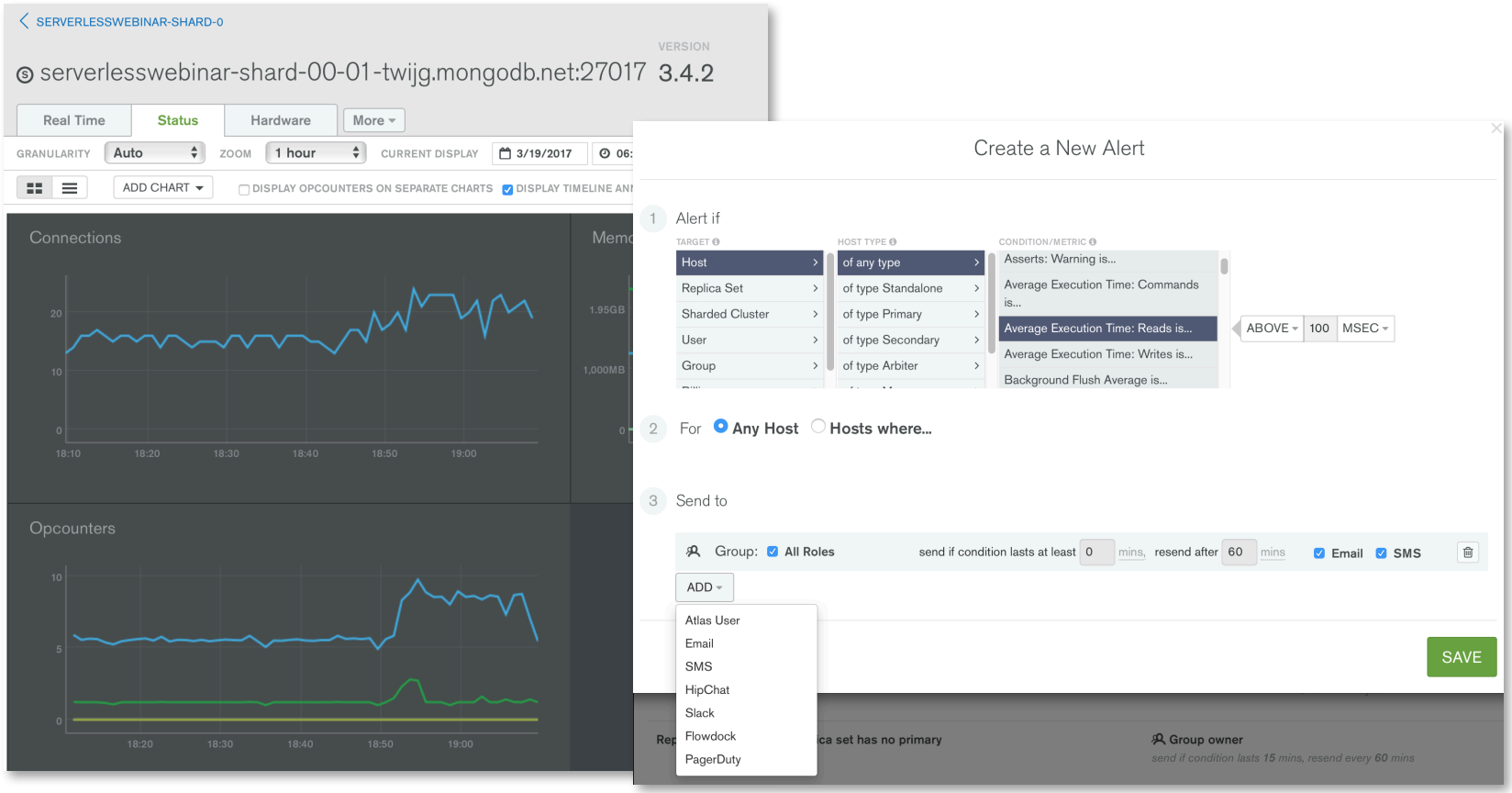
CloudWatch > Log Groups > /aws/lambda/GET_lambda_handler > 2017/03/21/[$LATEST]f5fc39dc71f647ea9b3d9f41753fe4ce

Expand all Row Text

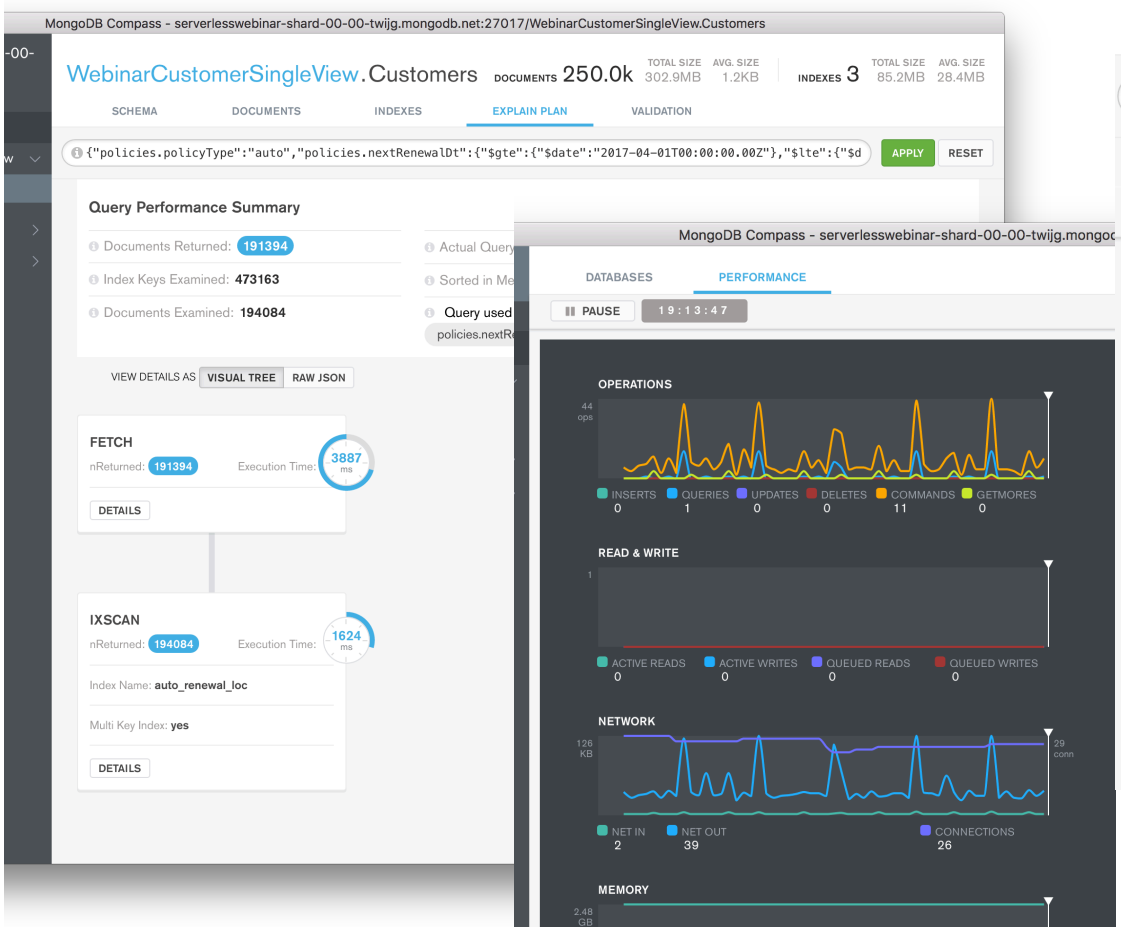
Filter events | all 30s 5m 1h 6h 1d 1w custom

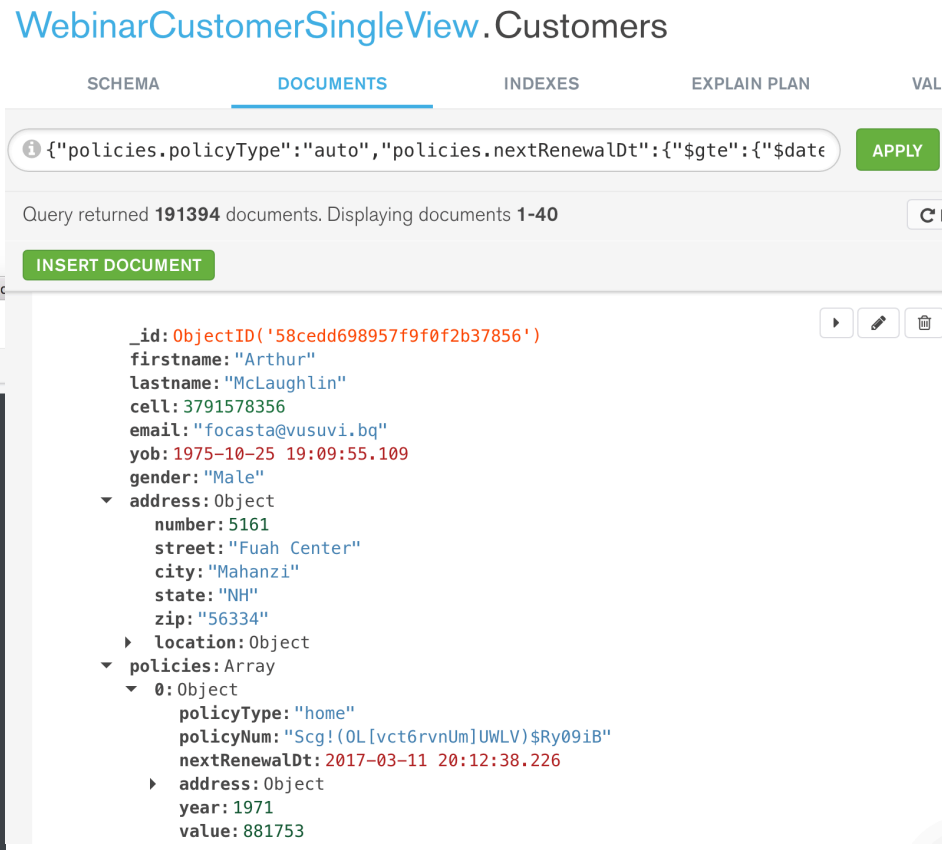
Time (UTC +00:00)	Message
2017-03-21	
	No older events found at the moment. Retry.
00:13:35	Loading function
00:13:35	=== CONNECTING TO MONGODB ATLAS ===
00:13:35	=== DONE - DB NAME: WebinarCustomerSingleView ===
00:13:35	START RequestId: 31fd10bf-0dcb-11e7-928e-518cbd547b30 Version: $LATEST
00:13:35	filter = {}
00:13:36	END RequestId: 31fd10bf-0dcb-11e7-928e-518cbd547b30
00:13:36	REPORT RequestId: 31fd10bf-0dcb-11e7-928e-518cbd547b30 Duration: 735.96 ms Billed Duration: 800 ms Memory Size: 128 MB Max Memory Used: 26
00:13:36	START RequestId: 3a2a91f8-0dcb-11e7-9ccc-4ffa9bba1c06 Version: $LATEST
00:13:36	filter = {"_id": {"$oid": "58cedd698957f9f0f2b37863"}}
00:13:36	END RequestId: 3a2a91f8-0dcb-11e7-9ccc-4ffa9bba1c06
00:13:36	REPORT RequestId: 3a2a91f8-0dcb-11e7-9ccc-4ffa9bba1c06 Duration: 3.98 ms Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 26 MB
00:13:40	START RequestId: 3c59ce72-0dcb-11e7-9592-85950a4fcfc2 Version: $LATEST
00:13:40	filter = {}
00:13:40	END RequestId: 3c59ce72-0dcb-11e7-9592-85950a4fcfc2
00:13:40	REPORT RequestId: 3c59ce72-0dcb-11e7-9592-85950a4fcfc2 Duration: 511.89 ms Billed Duration: 600 ms Memory Size: 128 MB Max Memory Used: 26
00:13:40	START RequestId: 3ce17479-0dcb-11e7-8a1f-13c38df468a9 Version: $LATEST
00:13:40	filter = {"_id": {"$oid": "58cedd698957f9f0f2b37863"}}
00:13:40	END RequestId: 3ce17479-0dcb-11e7-8a1f-13c38df468a9
00:13:40	REPORT RequestId: 3ce17479-0dcb-11e7-8a1f-13c38df468a9 Duration: 2.34 ms Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 26 M
00:13:42	START RequestId: 3dea58ec-0dcb-11e7-b370-afe0f068a543 Version: $LATEST
00:13:42	filter = {}
00:13:43	END RequestId: 3dea58ec-0dcb-11e7-b370-afe0f068a543
00:13:43	REPORT RequestId: 3dea58ec-0dcb-11e7-b370-afe0f068a543 Duration: 329.55 ms Billed Duration: 400 ms Memory Size: 128 MB Max Memory Used: 26
00:13:43	START RequestId: 3e4571db-0dcb-11e7-8d1e-e79450dc4d9c Version: $LATEST
00:13:43	filter = {"_id": {"$oid": "58cedd698957f9f0f2b37863"}}

MongoDB Atlas Monitoring and Alerts



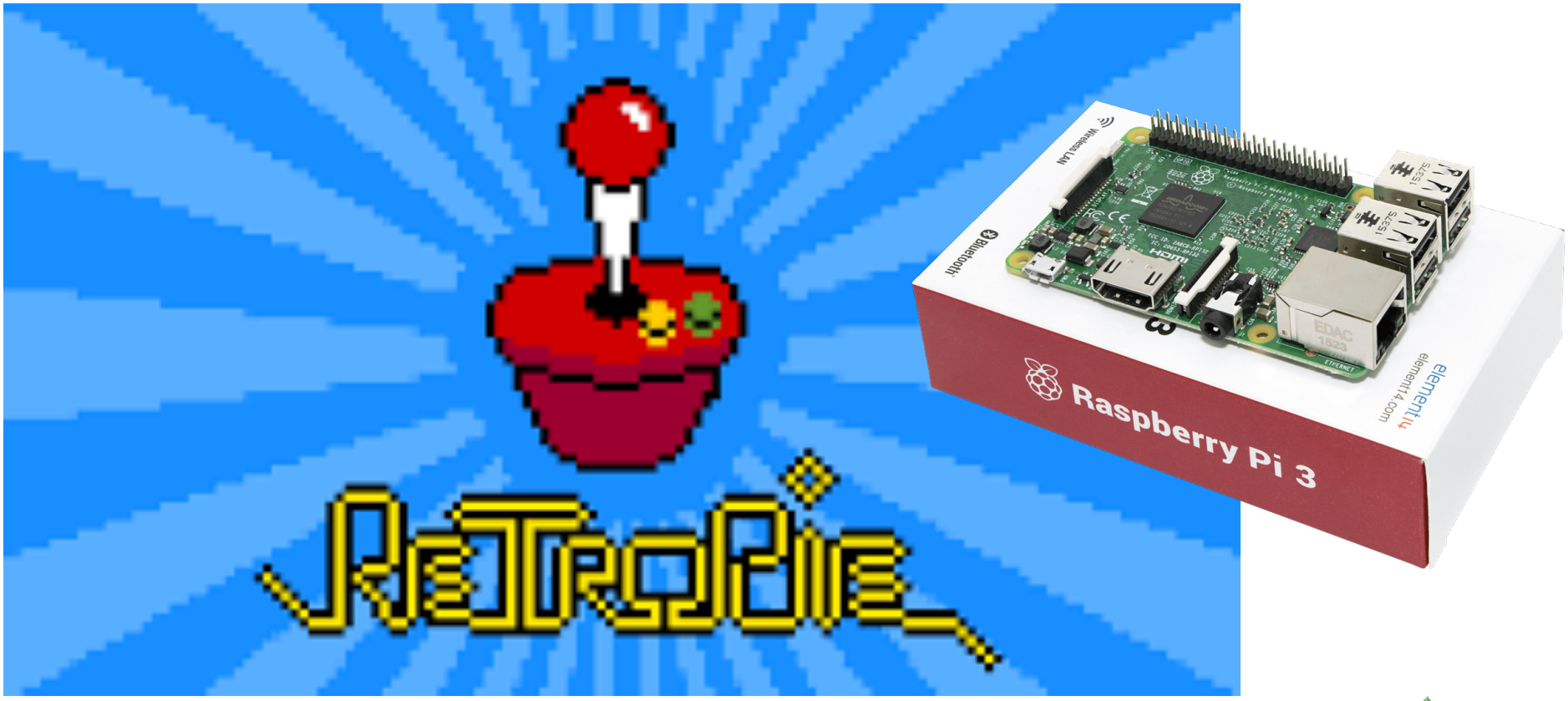
MongoDB Compass



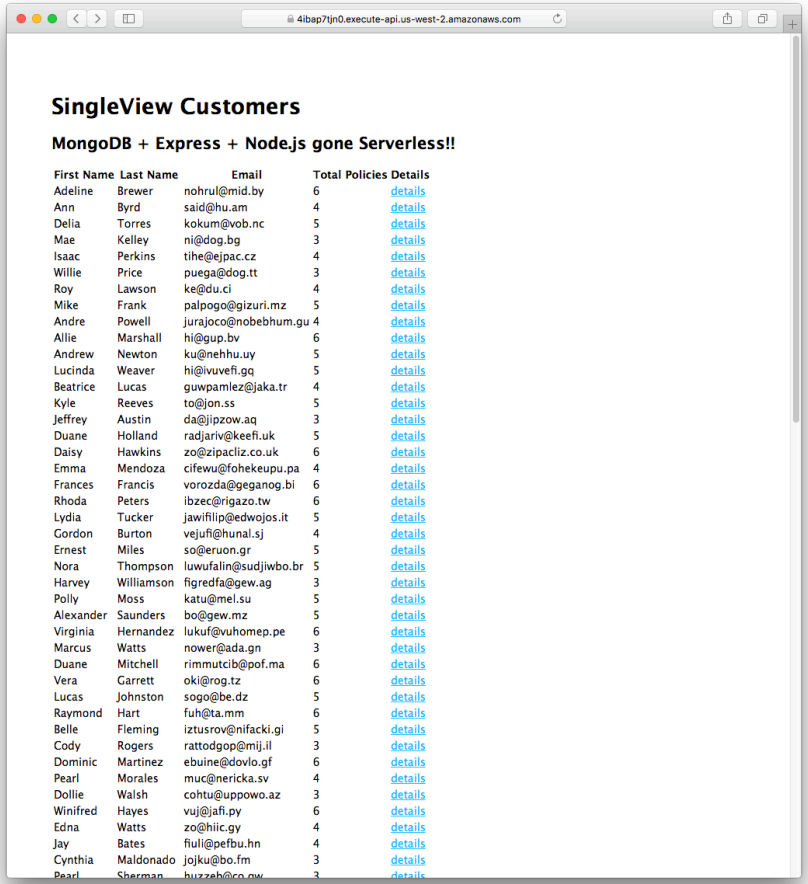


Serverless Web too!!

RetroPie – Old-school games, killer tiny hardware



Retro Web App based on aws-serverless-express



https://github.com/snarvaez/bddla17/tree/master/singleViewWebApp

https://github.com/awslabs/aws-serverless-express

mongoDB

Done!

...

But what about?

Scaling?

Scaling Lambda

No user intervention required - Default safety throttle of 100 concurrent executions per **account** per region.

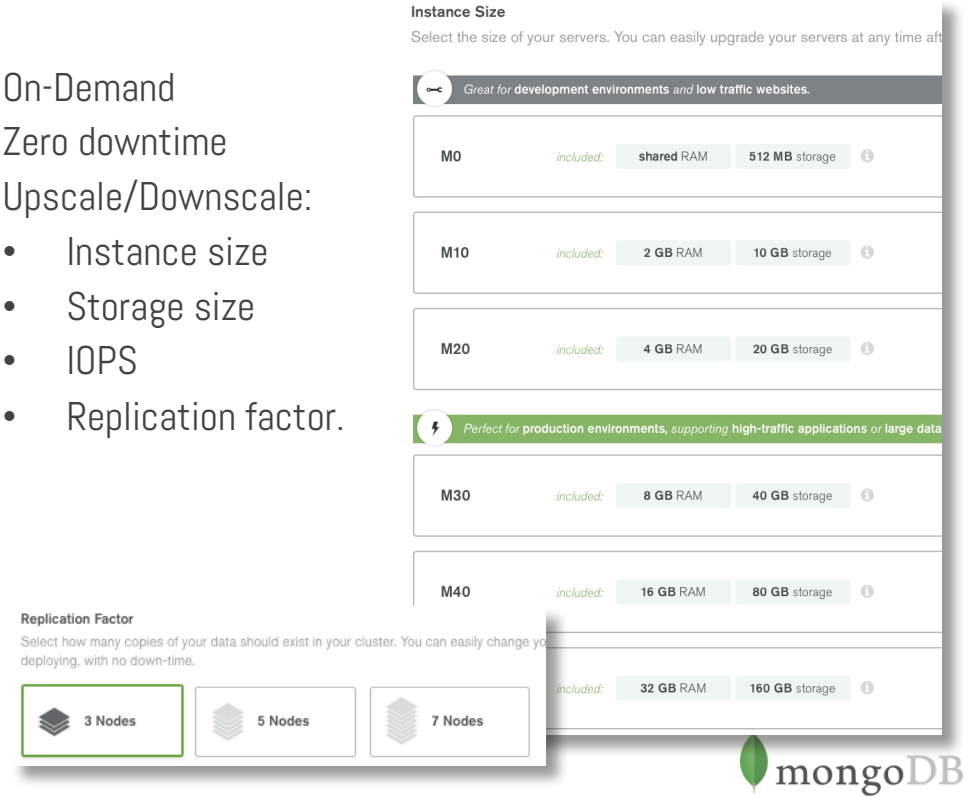
Functions invoked **synchronously** throw 429 error code. Functions invoked **asynchronously** can absorb reasonable bursts for approx. 15-30 minutes. If exhausted, consider using Simple Queue Service (SQS) or Simple Notification Service (SNS) as the Dead Letter Queue (DLQ).

Read more at https://aws.amazon.com/lambda/faqs/

Scaling MongoDB Atlas

On-Demand
Zero downtime
Upscale/Downscale:

- Instance size
- Storage size
- IOPS
- Replication factor.



Serverless Architectures with AWS Lambda and MongoDB Atlas

Q & A

Sig Narváez
Sr. Solutions Architect
sig@mongodb.com
@SigNarvaez

https://github.com/snarvaez/bddla17

https://resources.mongodb.com/serverless-architectures