

ARC303

AWS re:INVENT

Running Lean Architectures: How to Optimize for Cost Efficiency

Markus Ostertag, VP of Engineering, Team Internet AG, @osterjour
Constantin Gonzalez, Principal Solutions Architect, AWS, @zalez

November 28, 2017

AWS re:INVENT



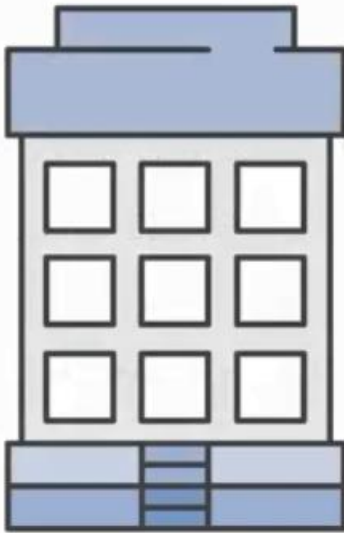
Whether you're a cash-strapped startup or an enterprise optimizing spend, it pays to run cost-efficient architectures on AWS. This session reviews a wide range of cost planning, monitoring, and optimization strategies, featuring real-world experience from AWS customers. We cover how to effectively combine Amazon EC2 On-Demand, Reserved, and Spot Instances to handle different use cases; leveraging Auto Scaling to match capacity to workload; and choosing the optimal instance type through load testing. We discuss taking advantage of tiered storage and caching, offloading content to Amazon CloudFront to reduce back-end load, and getting rid of your back end entirely by serverless. Even if you already enjoy the benefits of serverless architectures, we show you how to select the optimal AWS Lambda memory class and how to maximize networking throughput in order to minimize Lambda run-time and therefore execution cost. We also showcase simple tools to help track and manage costs, including Cost Explorer, billing alerts, and AWS Trusted Advisor. This session is your pocket guide for running cost effectively in the AWS Cloud.

What you'll get out of this session

- Best practices on how to lower your AWS bill
- A more scalable, robust, dynamic architecture
- More time to innovate
- Real-world customer examples
- Easy to implement



Structure



Business
Architecture
Operations



AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Business goals

"Pay as little
as possible
for what we
use."

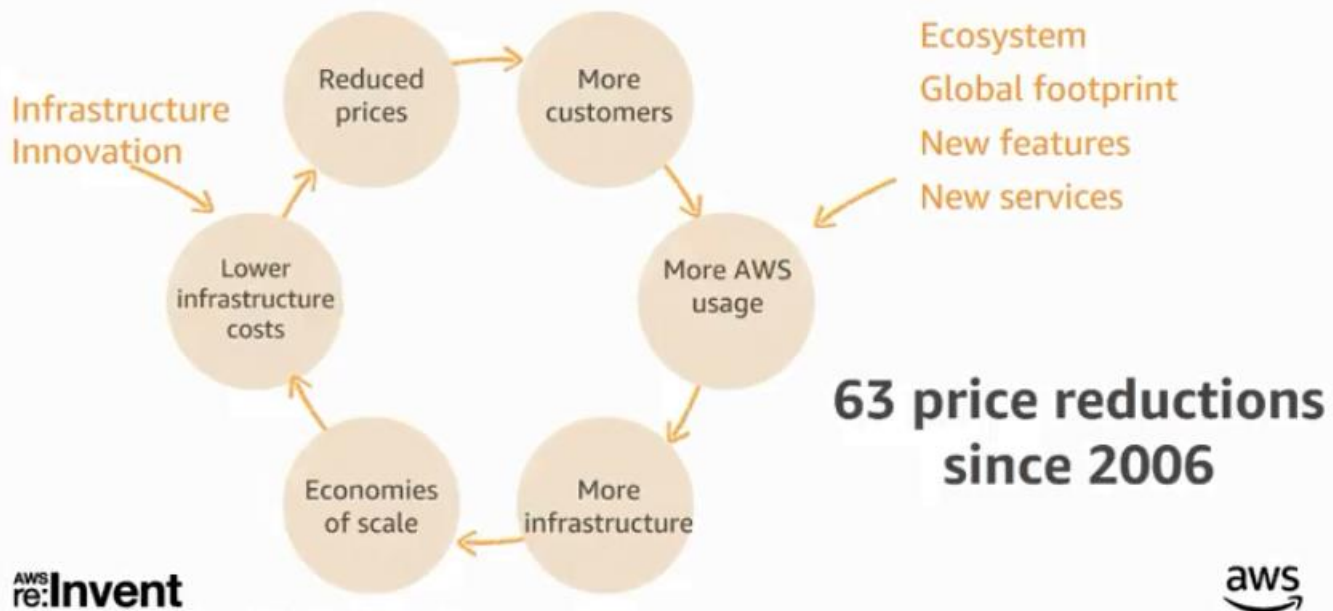


AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



AWS pricing philosophy



AWS TCO calculator



aws
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

<https://awstcocalculator.com/>

aws

TCO Calculator

Secure | https://awstcocalculator.com

[Contact Sales](#)

AWS Total Cost of Ownership (TCO) Calculator

Basic

Use this calculator to compare the cost of running your applications in an on-premises or colocation environment to AWS. Describe your on-premises or colocation configuration to produce a detailed cost comparison with AWS. You can switch between the basic and advanced views to provide additional configuration details.

Select Currency

United States Dollar

What type of environment are you comparing against?

☒ On-Premises
 ☐ Colocation

Which AWS region is ideal for your geo requirements?

US East (N. Virginia)

Choose workload type:

General

Servers

Are you comparing physical servers or virtual machines?

☐ Physical Servers ☒ Virtual Machines

Provide your configuration details:

Server Type	App. Name	Number of VMs	CPU Cores	Memory(GB)	Hypervisor	Guest OS	DB Engine
Non DB		1 - 10000	1 - 32	1 - 256	VMware	Linux	

Total no.of VMs:

+ Add Row

Storage

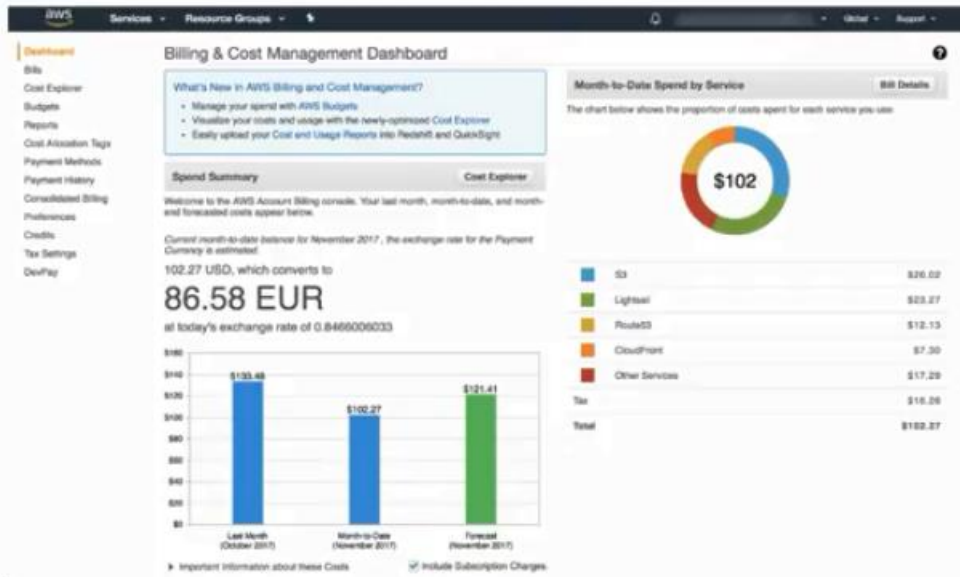
Provide your storage footprint details

Storage Type	Raw Storage	% Accessed
--------------	-------------	------------



You should also set up your AWS Billing alarms for your services

AWS billing dashboard



AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



AWS cost explorer—now with API!



AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Markus Ostertag
VP of Engineering @ Team Internet

Who is Team Internet?

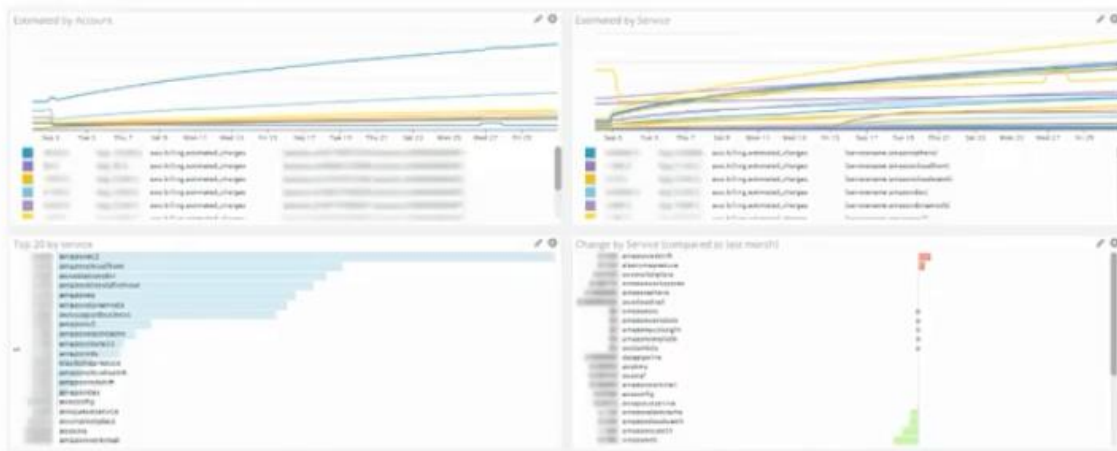
- Domain monetization business
- 35 people
- HQ in Munich, Germany
- Tech focused



© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Diving deep into your bill using Datadog



Monitor and use your billing metrics!



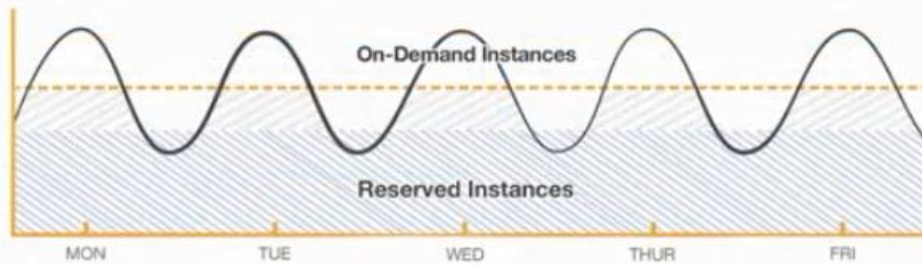
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



We have cost break down by accounts and services used in AWS.

Reserved Instances

Save up to 60% (up to 40% with one year)



Convertible RI

Change instance family, OS, tenancy

One year or three years

AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Reserved Instances—our usage hours



Baseline backed by Reserved Instances

Excess with On-Demand or Spot

New instance size flexibility for RIs

Within the same family **regional Linux/UNIX Reserved Instances with shared tenancy** are instance size flexible now

Example:

c4.4xlarge RI counts for

2x c4.2xlarge,
4x c4.xlarge,
8x c4.large, or
0.5x c4.8xlarge

Instance Size	Normalization Factor
nano	0.25
micro	0.5
small	1
medium	2
large	4
xlarge	8
2xlarge	16
4xlarge	32
8xlarge	64
10xlarge	80
16xlarge	128
32xlarge	256

AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Architecture goals

"Avoid waste
as much
as possible."



AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Turn off unused instances

- Developer, test, training instances
- Use simple instance start and stop
 - Even on Amazon Relational Database Service!
- Or tear down and build up all together using AWS CloudFormation
- Instances are disposable!

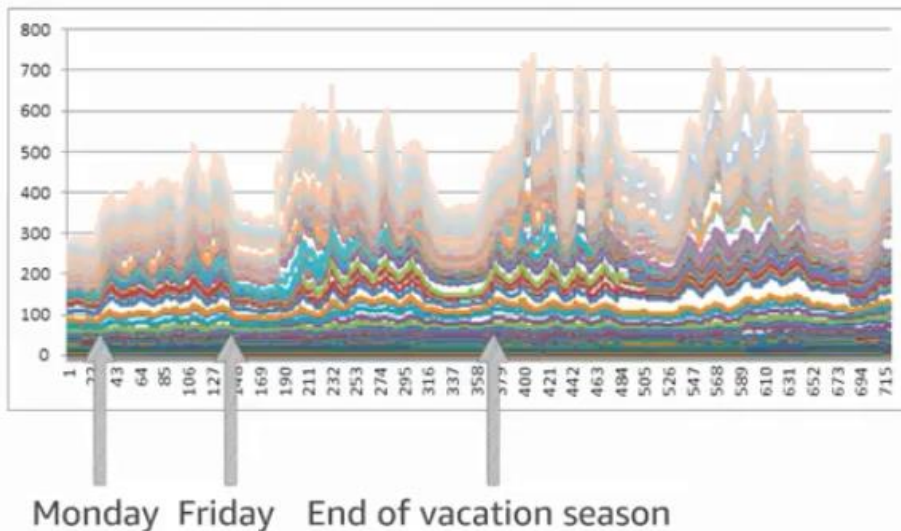


AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Customer example



35% saved

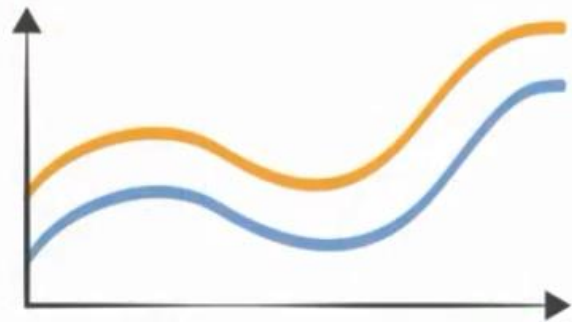
AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Automate, automate, automate

- AWS SDKs
- AWS Command Line Interface
- AWS CloudFormation
- AWS OpsWorks
- Netflix Janitor Monkey
- Cloudlytics EC2 Scheduler
- Auto Scaling



AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Auto Scaling is always possible!

Auto-Scaling groups are the goal

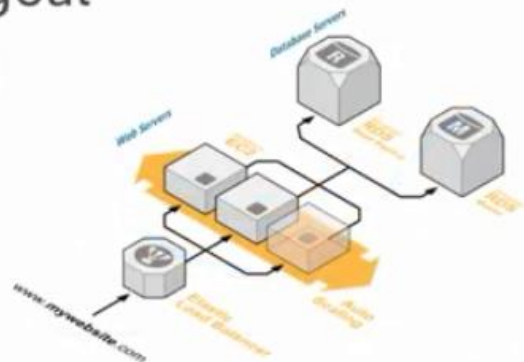
Side effect: cost optimization

Our to-dos:

Pet or cattle

Service discovery

Deployment strategy



AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Pets or cattle



AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



We need to automate every step from starting up our server and deleting them

Pets or cattle

User data:

```
#cloud-config
bootcmd:
[...]
```

```
AZ_NAME=`ec2metadata --availability-zone | sed -e 's/^.*-[0-9]//';
INSTANCE_ID=`ec2metadata --instance-id | sed -e 's/^i-//';
AWS_HOSTNAME=$HOSTNAME_PREFIX-'$AZ_NAME'-'$INSTANCE_ID';
echo $AWS_HOSTNAME > /etc/hostname;
hostname -F /etc/hostname;
aws ec2 create-tags --resources `ec2metadata --instance-id` --tags
Key=Name,Value=$AWS_HOSTNAME --region us-east-1
```

Leverage user data and cloud-init or cfn-init (for AWS CloudFormation)

AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



You can use the script to also name your servers in an automated fashion

Service discovery

- Use tools like Consul, Netflix Eureka
- Or build your own:
 - Scheduled AWS Lambda function (using tags)
 - Put A-Records in Route53 private zones



AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Deployment via cloud-init at instance launch

Using or mimicking AWS CodeDeploy
User data and AWS CLI:



```
#cloud-config
bootcmd:
  - aws s3 cp s3://<deploy-scripts-bucket>/init/rtb.sh - | bash
```

RTB.sh:

- Get code from S3-bucket
- Copy to local EBS volume
- Start via systemd

AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Cost optimization with Auto Scaling

- Per second billing! *New*
 - You can **scale down faster** now
 - Focus on **performance**, less on cost
- Optimize cost through **dynamic** scale-in and scale-out
- Leverage **scheduled scaling events** to anticipate load changes
- Use Spot Instances



AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Use Spot Instances

- Price based on **supply/demand**
- You choose your **maximum** price/hour
- Your instance is started if the Spot price is **lower**
- Your instance is **terminated** if the Spot price is higher, with **two-minutes'** notice
- But: You can now choose **"stop"** instead of "terminate" *New*
- And: You did **plan for fault tolerance**, didn't you?



AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Spot Instance example



AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Spot Instance use cases

- Stateless Web/app server fleets
- Amazon EMR
- Continuous integration (CI)
- High performance computing (HPC)
- Grid computing
- Media rendering/transcoding



<https://aws.amazon.com/ec2/spot>

AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Spot Bid Advisor

Spot Bid Advisor

Region: US East (N. Virginia) OS: Linux/UNIX Bid Price: 50% On-Demand

Instance type filter:
vCPU (min): 1 Memory GB (min): 0 ☐ Instance types supported by EMR

Instance Type	vCPU	Memory GB	Savings over On-Demand ¹	Frequency of being outbid (month) ²	Frequency of being outbid (week)
c1.medium	2	1.7	80%	Low	Low
c1.xlarge	8	7	78%	Low	Low
c3.2xlarge	8	15	73%	Low	Low
c3.large	2	3.75	83%	Low	Low
c3.xlarge	4	7.5	78%	Low	Low
c4.2xlarge	8	15	76%	Low	Medium
c4.xlarge	16	30	78%	Low	Medium
c4.8xlarge	36	60	75%	Low	Medium
c4.large	2	3.75	77%	Low	Low
c4.xlarge	4	7.5	76%	Low	Low

Display all 72 instance types

AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Spot Instances recap

- **Dynamic** pricing
- Opportunity to **save** 80% to 90% cost
 - But be careful
- Different prices **per AZ**
- Leverage **Auto Scaling!**
 - One group with Spot Instances
 - One group with On-Demand
 - Get the best of both worlds
- Spot Fleet – Manage thousands of Spot Instances with one API call



AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



“But my applications are
too small
even for the smallest instance!”

Amazon EC2 Container Service

- Easily manage docker containers
- Flexible container placement
- Designed for use with other AWS services
- Extensible
- Performance at scale
- Secure



AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

Consolidate with Amazon ECS



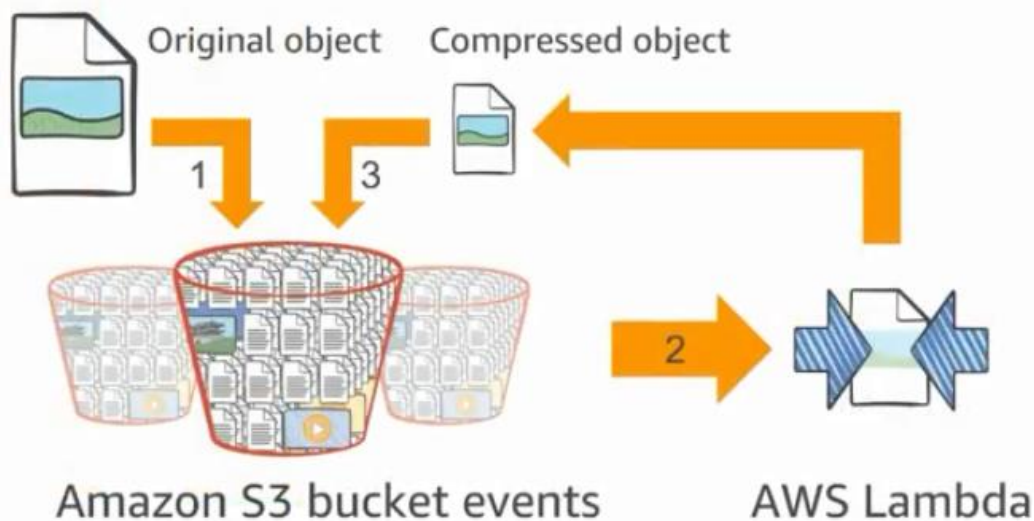
AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

When you have lots of small apps doing very short tasks, you need to leverage the ECS service. Try and find those single EC2 instances you are running that are doing some dumb simpler jobs but are still idling at 10% usage, then consolidate them all into a smaller number of instances running in ECS.

AWS Lambda and the serverless revolution



AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Get rid of idle time with AWS Lambda

- Automatic scaling
- Automatic provisioning
- No need to manage infrastructure
- Just bring your code
- \$0.20/million requests, 1M free per month
- 100 ms payment granularity
- Never pay for idle



**Less than 40% utilization?
Consider using Lambda instead!**

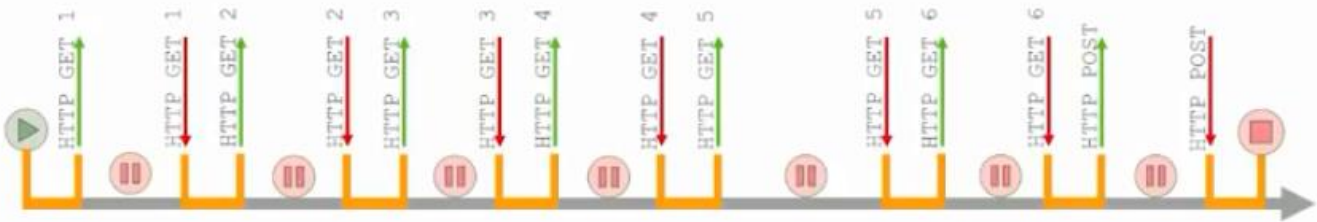
AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



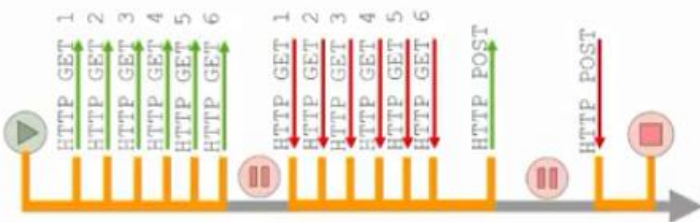
If your existing EC2 application is running lesser than 40% utilization, you will save money by porting that application to use lambda instead.

Beware of in-function idle time...



You also need to keep track of how much of your lambda function capacity is being used for computing and how much is being used for waiting? The grey areas are all waiting areas that you are paying for. You need to optimize this by parallelize your API calls

...use non-blocking code instead!



Use:

- Node.js events
- Java/Python/C# threads
- Coordinate with queues

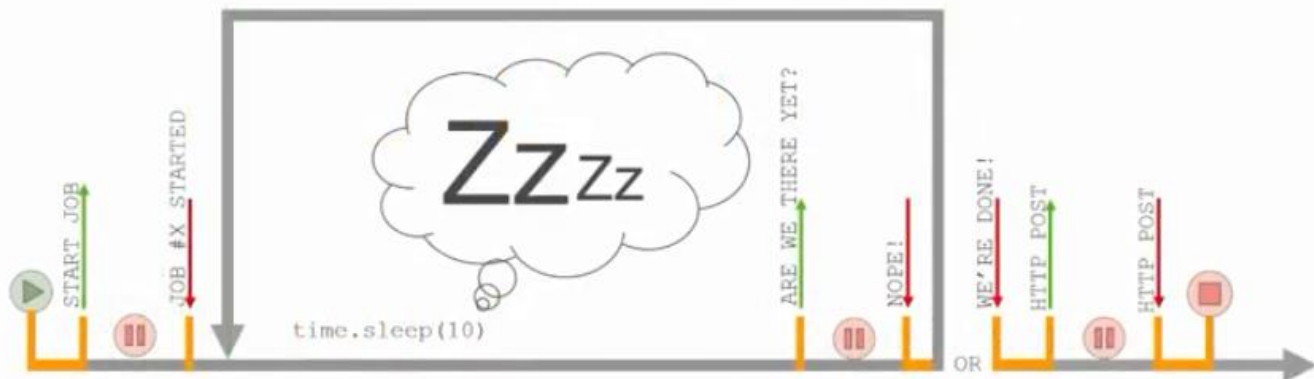
AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



You can now be done earlier with less waiting. You simply coordinate the API calls using threads and queues when available for multithreading

Avoid sleeping inside your function...



AWS re:Invent

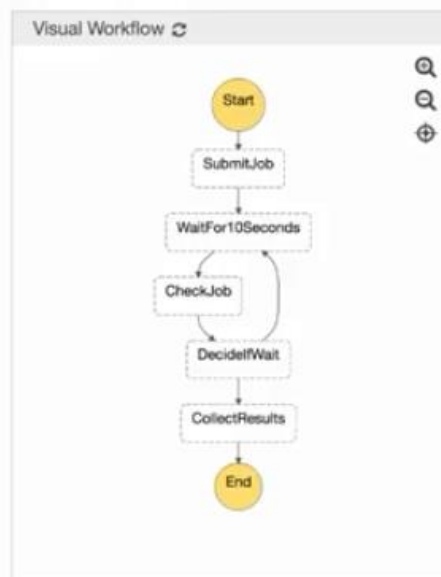
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



You need to remove `time.sleep()` functions and instead use AWS Step Functions

...use AWS Step Functions instead!

```
Code
1 {
2   "Comment": "An example of the Amazon States Language using wait states",
3   "StartAt": "SubmitJob",
4   "States": {
5     "SubmitJob": {
6       "Type": "Task",
7       "Resource": "arn:aws:lambda:REGION:ACCOUNT_ID:function:JOB_SUBMITTER",
8       "Next": "WaitFor10Seconds"
9     },
10    "WaitFor10Seconds": {
11      "Type": "Wait",
12      "Seconds": 10,
13      "Next": "CheckJob"
14    },
15    "CheckJob": {
16      "Type": "Task",
17      "Resource": "arn:aws:lambda:REGION:ACCOUNT_ID:function:JOB_CHECKER",
18      "Next": "DecideIfWait"
19    },
20    "DecideIfWait": {
21      "Type": "Choice",
22      "Choices": [
23        {
24          "Variable": "$status",
25          "StringEquals": "Done",
26          "Next": "CollectResults"
27        }
28      ],
29      "Default": "WaitFor10Seconds"
30    },
31    "CollectResults": {
32      "Type": "Task",
33      "Resource": "arn:aws:lambda:REGION:ACCOUNT_ID:function:RESULTS_COLLECTOR",
34      "End": true
35    }
36  }
37 }
```



AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Step functions allows you to place control logic in the cloud as a step function workflow where you can submit your job at the beginning of the workflow as a step, then you can implement a simple looping logic that will poll for the job to be completed, and if it is not completed it should do the waiting for you because waiting in step functions is for free. You don't pay for wait cycles in step functions, you can wait for seconds, hours, days, weeks, and even up to a year. You can simply decompose your lambda function into 2 simple lambdas to submit the job and another lambda that collects the results and processes it, you can save a lot of money

Coca-Cola saved 90s Lambda time... ...for each bottle sold!



<https://aws.amazon.com/blogs/aws/things-go-better-with-step-functions/>

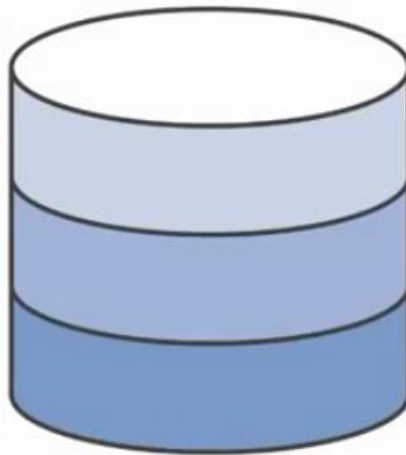
AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Coca cola has a system where when you buy a bottle from any of their vending machines, they will update your loyalty points account for you. Since the update of points functionality needs to wait for 90 seconds, they are using step functions for this and not having to wait in their lambda functions.

Optimizing database utilization

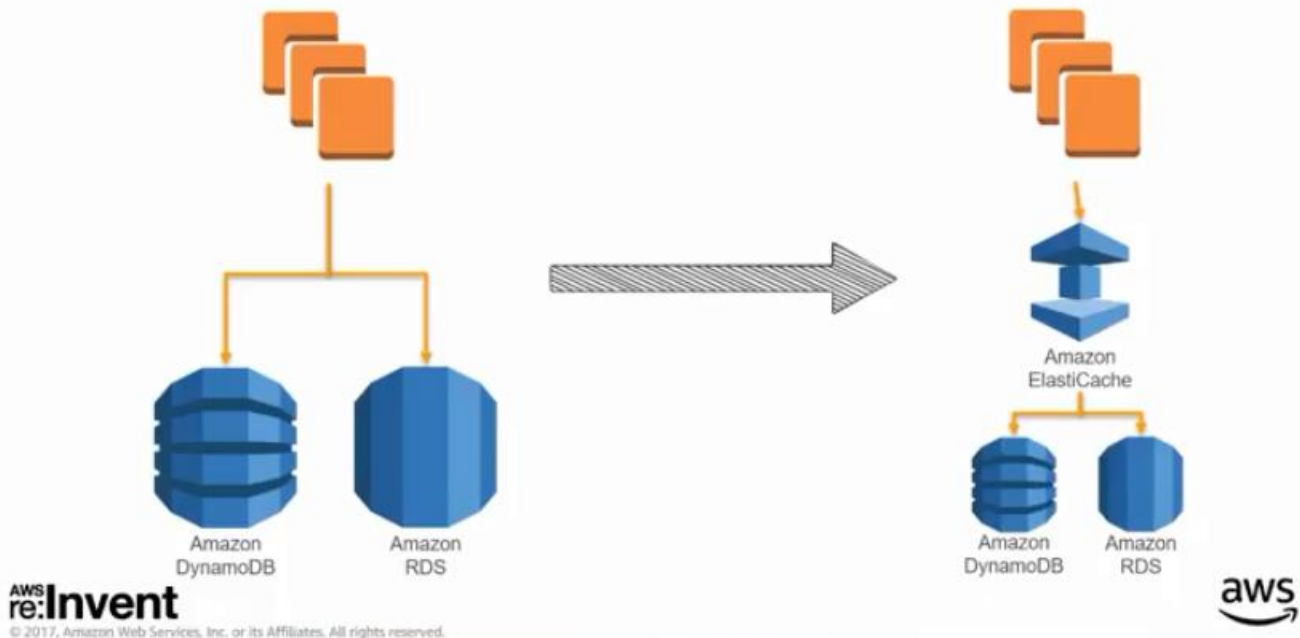


AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Database optimization through caching



Caching helps nearly all the time with your database.

Caching saves money

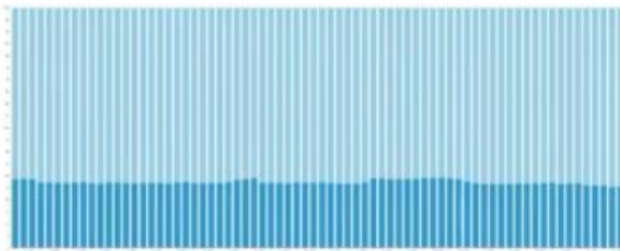


Saved 3K reads per second (> 20K reads per second in total)

By using a single Redis node for cache, we can reduce the needed DynamoDB read capacity that we need to provision

Cache really everything!

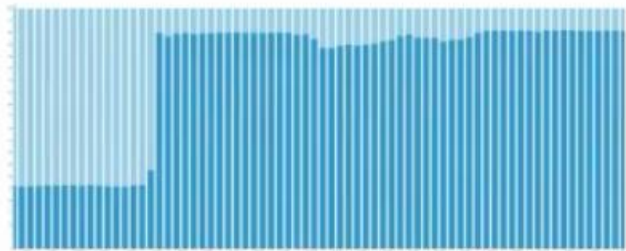
Without negative caching



Cache hit ratio 25% to 30%

Hit
Miss

With negative caching



Cache hit ratio 89% to 95%

AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Negative cache means we are saving the fact that there is no result for a certain query also to the Redis node instead of querying for it every time the query is made and we will still not get anything result

DynamoDB optimization

Think of strategies for **optimizing** CU use

- Use multiple tables to support varied access patterns
- Understand access patterns for time series data
- Compress large attribute values



Consider **read** (4K) against **write** (1K) sizes

Use Amazon Simple Queue Service to **buffer** overcapacity **writes**

Resize capacity units **dynamically**

AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



We also can optimize a lot of things in DynamoDB like the capacity usage for both reads and writes.

Use Auto Scaling for DynamoDB

Auto Scaling

☒ Read capacity

☒ Write capacity

☐ Same settings as read

Target utilization %

Minimum provisioned capacity units

Maximum provisioned capacity units

☒ Apply same settings to global secondary indexes

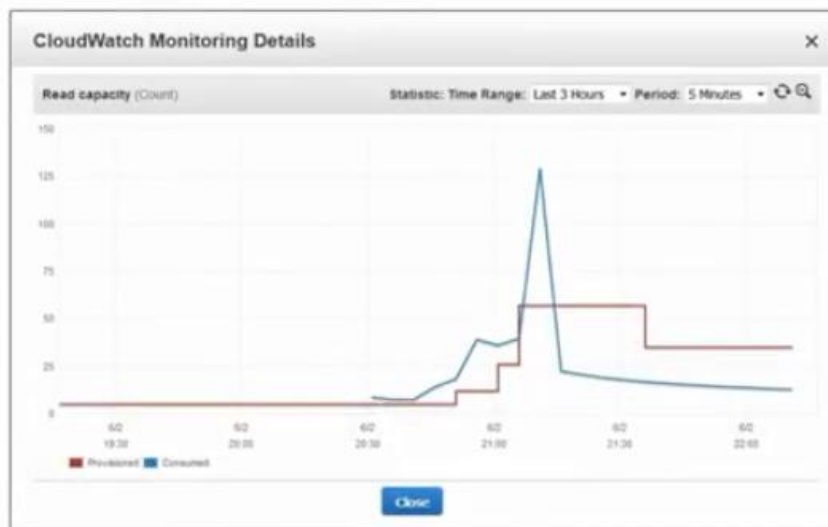
☒ Apply same settings to global secondary indexes

AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Use Auto Scaling for DynamoDB

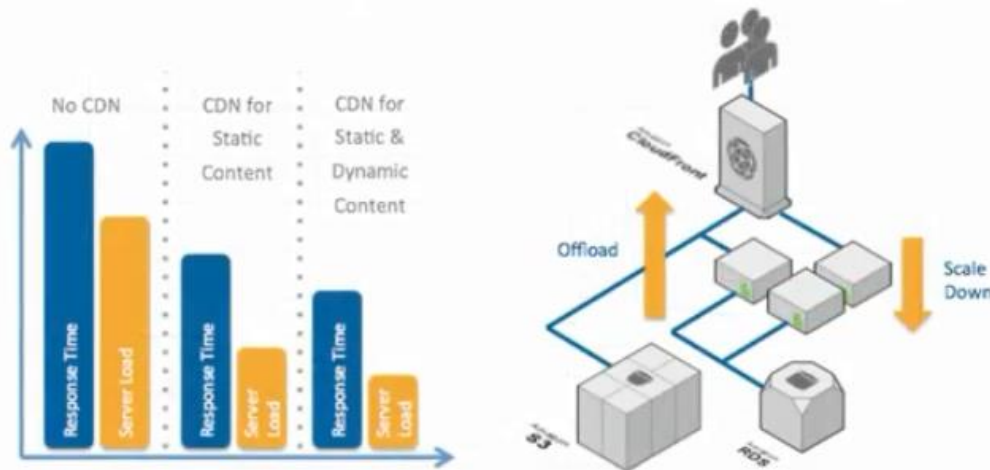


AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Offload popular traffic to Amazon S3 and/or Amazon CloudFront



AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



You can use CloudFront with a very small TTL like a TTL of 0.5 secs and you will still see some improvements

Operational goals

“Focus on what you do best, let AWS do the rest.”



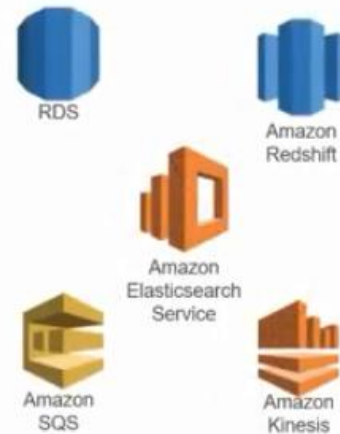
AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Leverage existing services

- Use Amazon RDS, DynamoDB, ElastiCache for Redis or Amazon Redshift
 - Instead of running your own database
- Amazon Elasticsearch Service
 - Instead of running your own cluster
- Amazon SQS
- Amazon Kinesis, Amazon Kinesis Firehose, Amazon Simple Notification Service, and more ...



AWS can help you with any service

AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



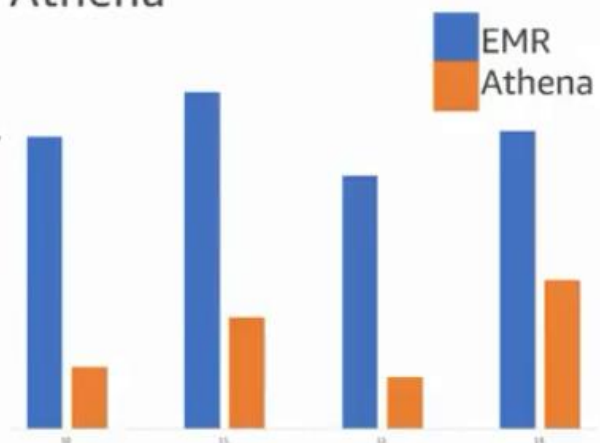
Amazon EMR to Amazon Athena migration

Our costs down by > 50% with Athena

Startup phase of EMR is crucial

Architecture simplicity

Less operations overhead

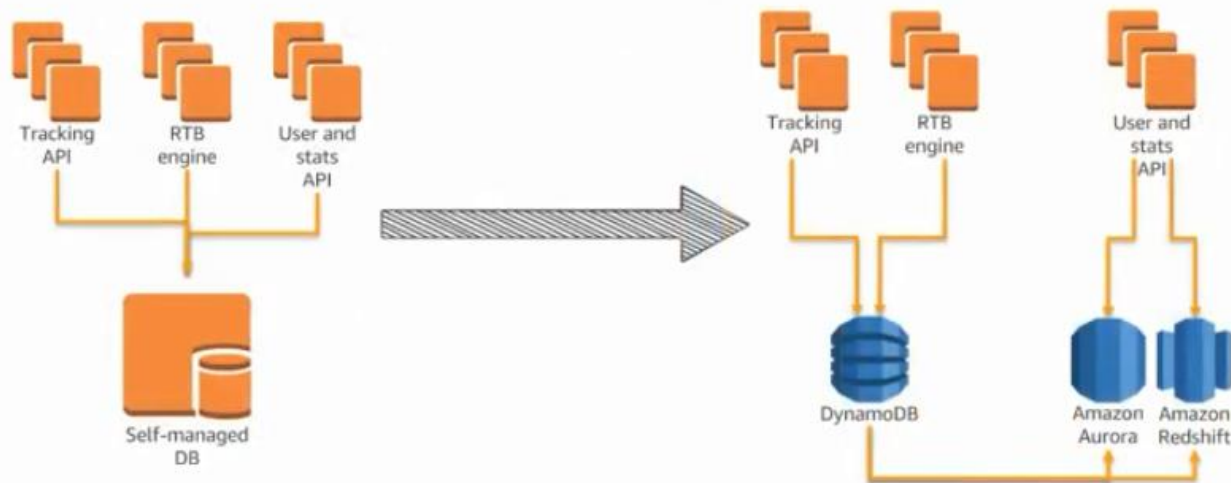


AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Split your architecture into several stacks

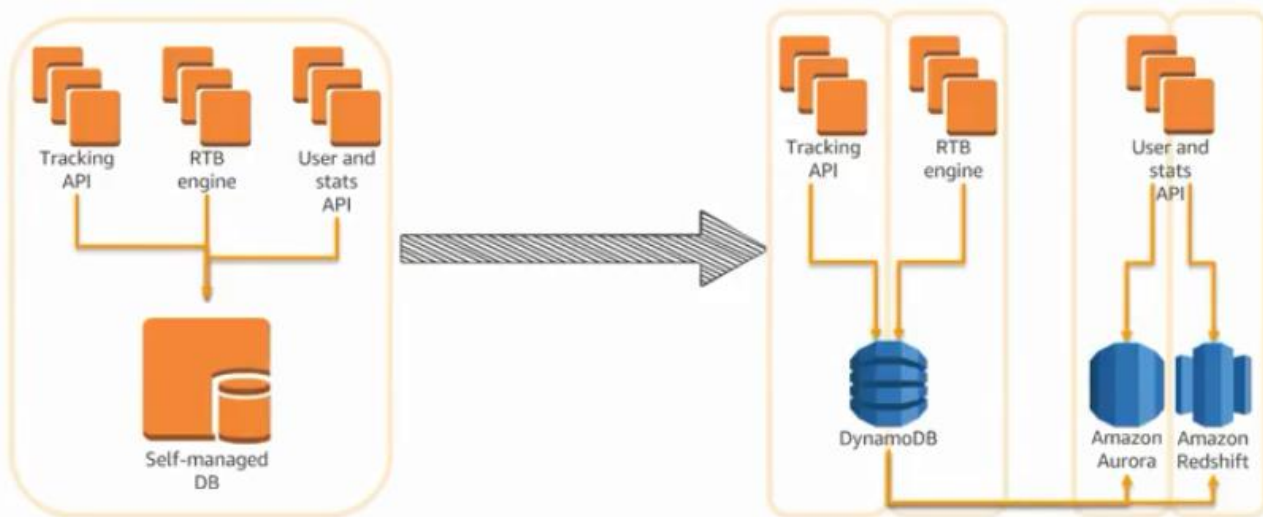


AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Split your architecture into several stacks



AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Our application architecture is now divided up into different stacks and the stacks now include the specific database it needs.

Pick the right tool for the job



DynamoDB

Key/value
Scalable
throughput
Low latency



Amazon Aurora

More
complex
data/queries
Scalable
storage



Amazon
Redshift

Big (complex)
data
Higher
latency



ElastiCache
for Redis

Key/Value
In-memory
(Very) low
latency

The challenge is now to find the best service for the specific job we have.

Every stack is an architecture

Divide the needs:

Write-heavy against read-heavy

Unstructured data against structured data

Consistent load against inconsistent load

Instance costs:

2x db.r3.large === 1x db.r3.xlarge

Advantages

- No undifferentiated heavy lifting (= saves money and work)
- AWS operates the DB infrastructure for us
- Simple and more granular scale out
- No interference between different functionalities/systems
- Issue pinpointing and team responsibility easier



AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Let's recap

1. Use AWS TCO/cost/billing tools
2. Use Reserved Instances
3. Avoid idle instances through automation
4. Use Spot Instances
5. Optimize database utilization
6. Pick the right tool for the job
7. Offload your architecture

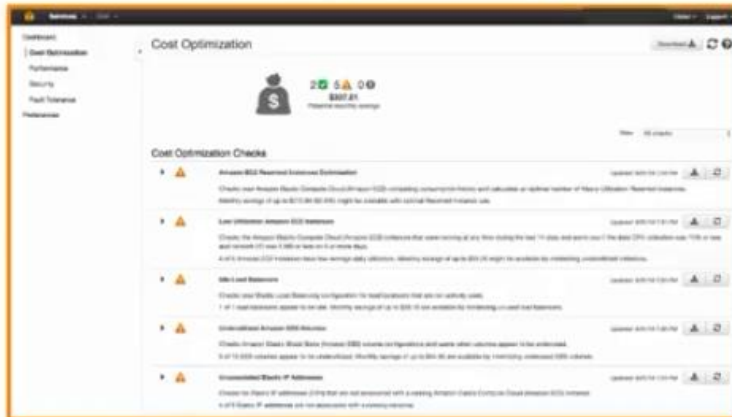


AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

AWS Trusted Advisor

aws.amazon.com/premiumsupport/trustedadvisor/



Included with business or enterprise support

AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



But wait! There's more...



<https://youtu.be/SG1DsYgeGEk>

AWS re:Invent 2015

ARC302 Running Lean Architectures:
Optimizing for Cost Efficiency



<https://youtu.be/pFpv6FsCVfY>

AWS re:Invent 2016

ARC313 Running Lean Architectures:
Optimizing for Cost Efficiency

AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



AWS re:Invent

Thank you!

markus@teaminternet.com | @osterjour
glez@amazon.de | @zalez

AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

