AWS Serverless Examples

AWS Serverless Components

AWS Serverless - Previous Slides Recap

- In previous slides we analyzed AWS main serverless components (https://www.slideshare.net/DimosthenisBotsaris/aws-serverless-introduction
- In these slides we will **explore serverless architecture flows** to solve real life issues.
- We try to define AWS Serverless components to use to achieve serverless flow, in production environment.
- **Implementation** of some flows can be found on: https://github.com/arconsis/aws-network-microservices-warmup

Serverless Architectures

Pros

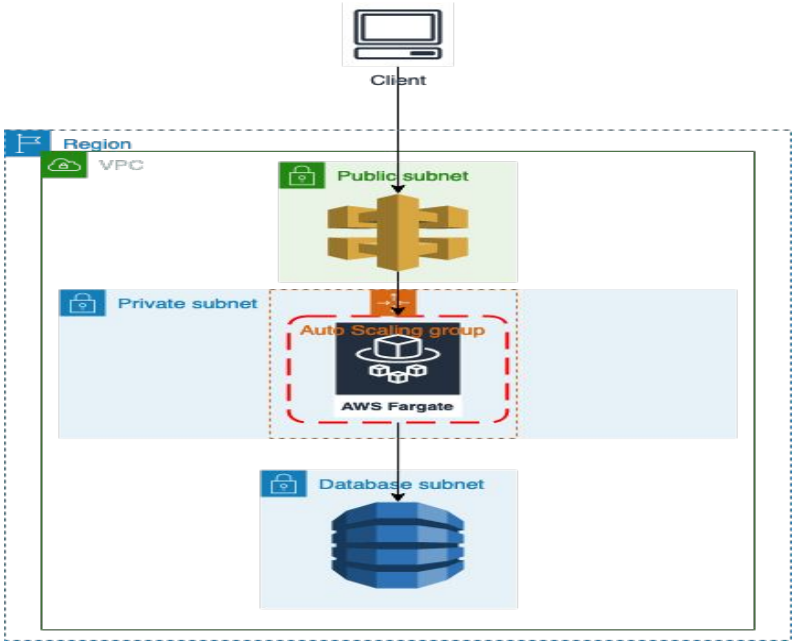
- ***Cost***: Pay per invocation, no costs for unused servers
- ***Scaling***: Auto scaling in response to spikes in traffic
- ***Productivity***: Responsible only for your code, AWS handles managing and provisioning of servers

Cons

- ***Vendor lock-in***: Seamlessly integration with other services from AWS, hard to move over to other Cloud Providers
- ***Testing***: Difficult to perform integration tests
- ***Performance***: Cold starts may add latency to some users

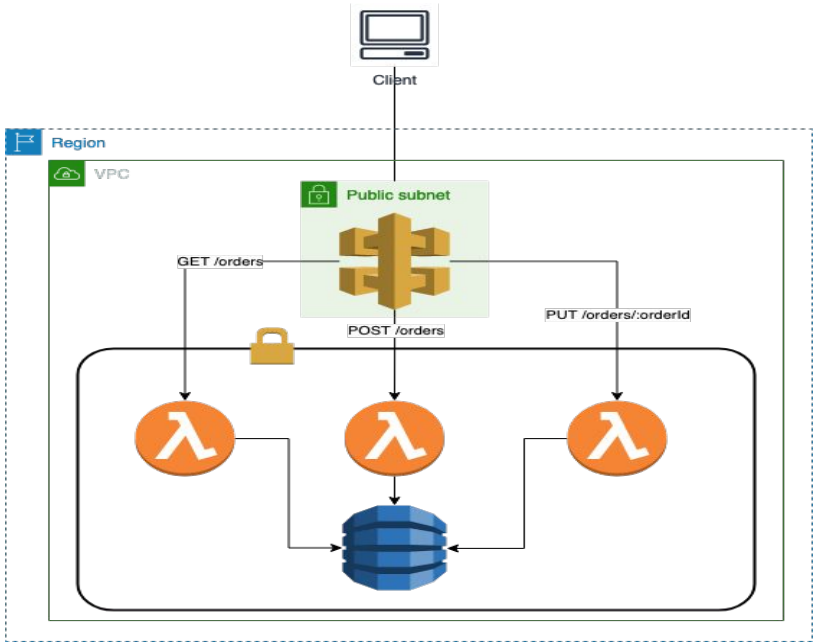
3-tier Serverless Flow (1)

- **Amazon API GW** in a public subnet, as entry point of backend.
- **Amazon API GW** handles routing, aggregation, authorization.
- **Amazon ECS** in private subnets handles orchestration management & auto-scale of our backend.
- **Amazon Fargate** in private subnets used as our server.
- **Amazon DynamoDB** in private subnets is used as database.



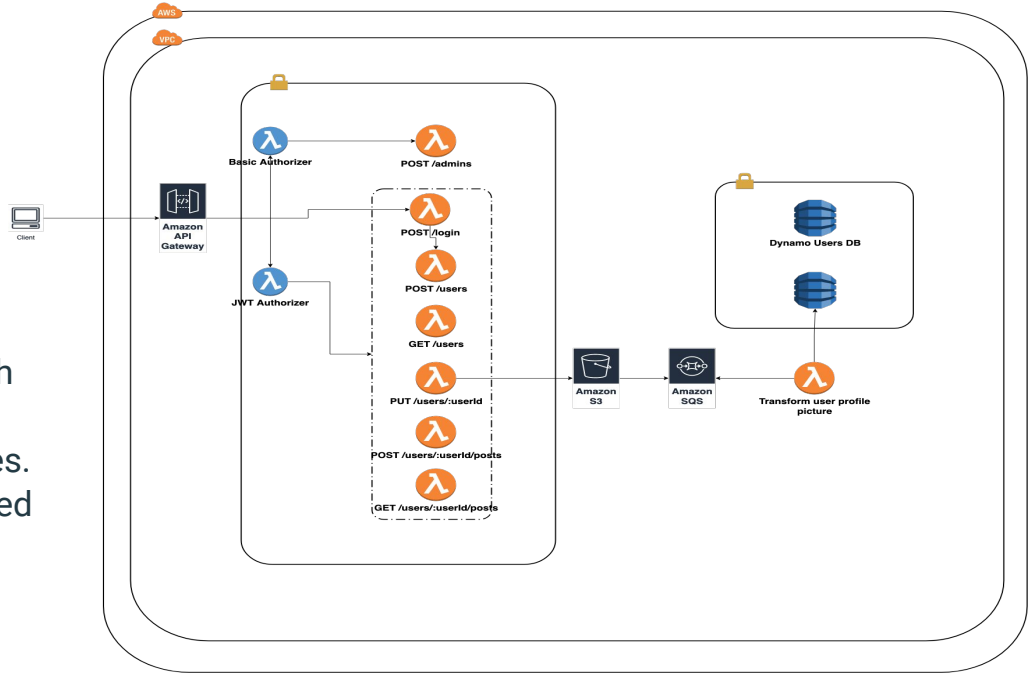
3-tier Serverless Flow (2)

- **Amazon API GW** in a public subnet, as entry point of backend.
- **Amazon API GW** handles routing, aggregation, authorization.
- **Amazon Lambda** in private subnets, used as servers, which auto-scale.
- **Amazon DynamoDB** in private subnets is used as database.

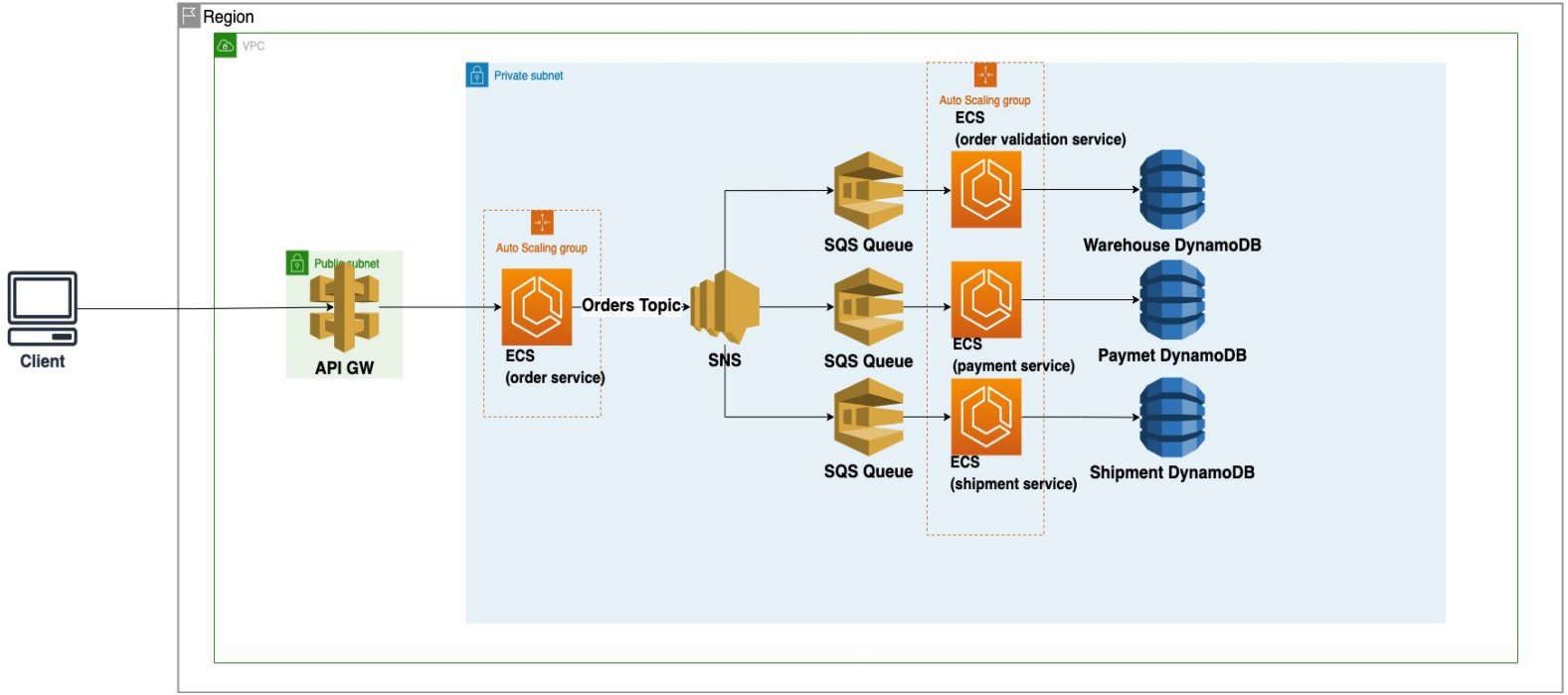


3-tier Serverless Flow (3)

- **Amazon API GW** in a public subnet, as entry point of backend.
- **Amazon API GW** handles routing, aggregation, authorization.
- **Amazon Lambda** in private subnets, used as servers, which auto-scale.
- **Amazon S3** is used to store files.
- **S3 event notification** will be fired towards SQS.
- **Amazon DynamoDB** in private subnets is used as database.



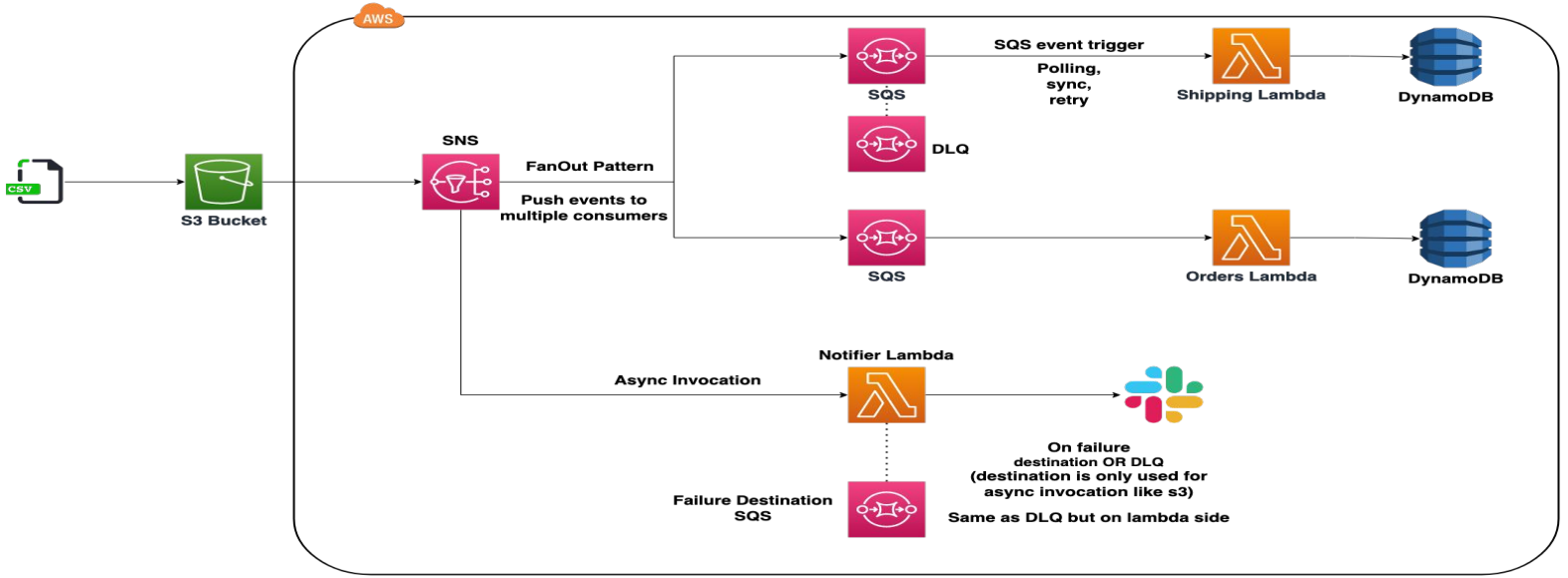
AWS Microservices Event Driven Flow (1)



AWS Microservices Event Driven Flow (2)

- **Amazon API GW** in a public subnet, as entry point of backend.
- **Amazon API GW** handles routing, aggregation, authorization.
- **Amazon ECS** in private subnets handles orchestration management & auto-scale of our backend.
- **Amazon Fargate** in private subnets used as our servers.
- **Amazon SNS + SQS** combination is used to create **Fan-Out** pattern, to serve an event to two or more downstream services.
- The **SQS queue** stores the event for asynchronous processing
- **Amazon DynamoDB** in private subnets is used as databases.

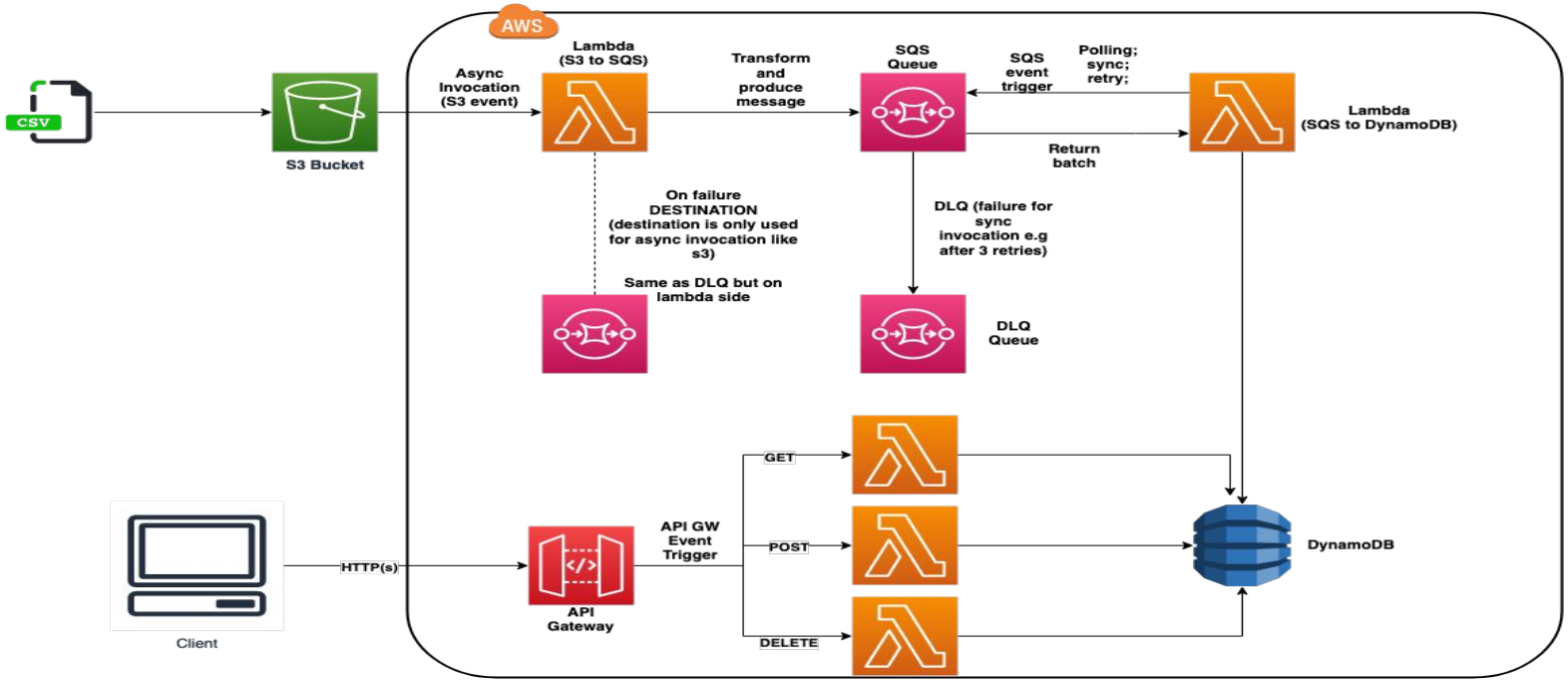
AWS S3 Events - FanOut (1)



AWS S3 Events - FanOut (2)

- **Amazon S3** is used as file storage.
- Client uploads a file, which is stored to AWS S3.
- **S3 event notification** will be fired towards SNS.
- **Amazon SNS + SQS** combination is used to create **Fan-Out** pattern, to serve an event to two or more downstream services.
- **Amazon SQS DLQ** is used for messages that can't be processed (consumed) successfully.
- The **SQS queue** stores the event for asynchronous processing.
- **Amazon Lambda** in private subnets, used as servers, which auto-scale.
- **Amazon DynamoDB** in private subnets is used as databases.

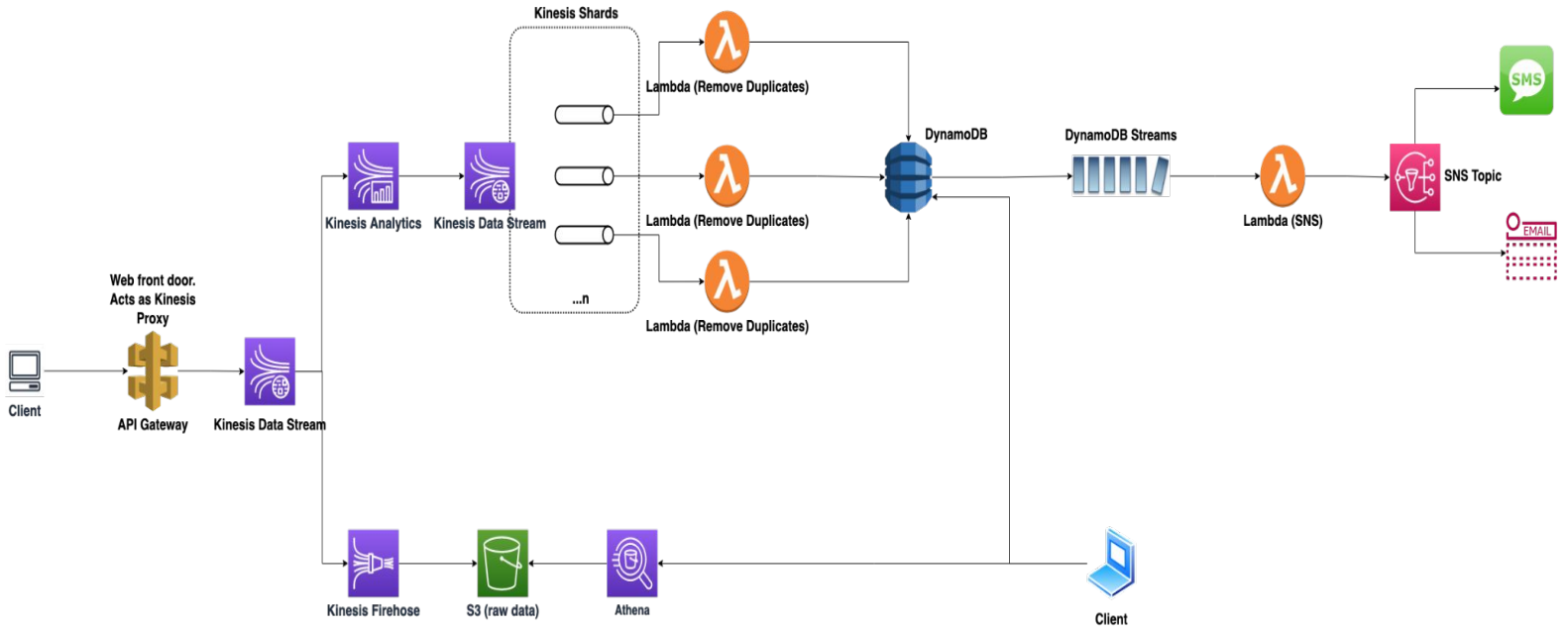
AWS S3 Events - Lambda (1)



AWS S3 Events - Lambda (2)

- **Amazon S3** is used as file storage.
- Client uploads a file, which is stored to AWS S3.
- **S3 event notification** will be fired towards **Lambda**.
- **Amazon SQS** is used for communication among Lambdas - decouple them!
- **Amazon SQS DLQ** is used for messages that can't be processed (consumed) successfully.
- The **SQS queue** stores the event for asynchronous processing.
- **Amazon API GW** in a public subnet, as entry point of backend.
- **Amazon API GW** handles routing, aggregation, authorization.
- **Amazon Lambda** in private subnets, used as servers, which auto-scale.
- **Amazon DynamoDB** in private subnets is used as databases.

AWS Real Time Analytics Stream Platform (1)

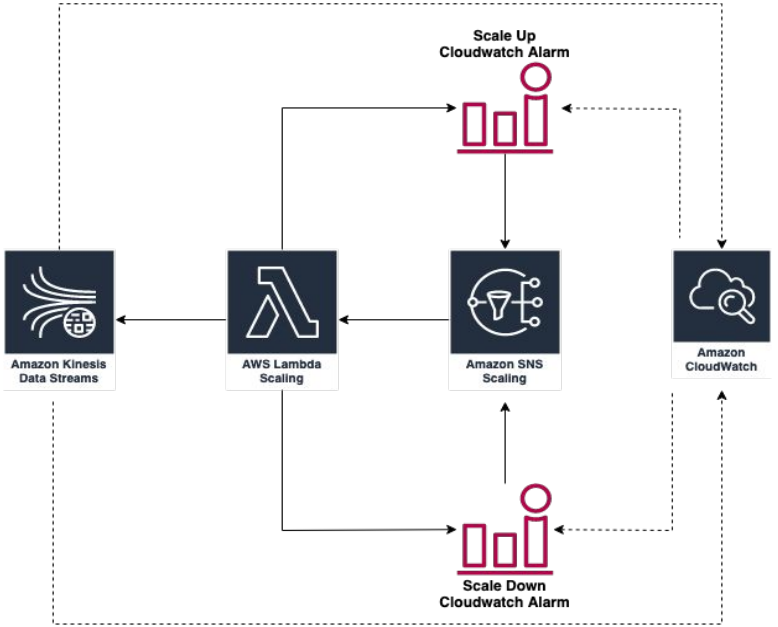


AWS Real Time Analytics Stream Platform (2)

- **Amazon API GW** in a public subnet, as entry point of backend and acts as Kinesis Proxy (decouple client from Kinesis)
- **Amazon Kinesis Data Stream** ingests and collects large amount of data records in real time.
- **Amazon Kinesis Data Analytics** used to transform and analyze streaming data in real time.
- **Amazon Lambda** in private subnets, used to add our business logic e.g. remove duplicate values.
- **Amazon DynamoDB** in private subnets is used as databases to store processed data.
- **Amazon DynamoDB streams** used to fire transactional logs events.
- **Amazon SNS** used to send sms / emails to clients.
- **Amazon Kinesis Data Firehose** used as ETL service that streams data into Amazon S3 into correct format.
- **Amazon S3** used to store raw data.
- **Amazon Athena** used interactive query service to query / analyze data from S3

AWS Kinesis AutoScale

- **Amazon CloudWatch Metrics** used to capture metrics from Kinesis Data Stream.
- **Amazon CloudWatch alarms** (scale-up/down), used to decide when to scale.
- When scaling takes place, an event fired to **Amazon SNS.**
- **Amazon Lambda** async consumes SNS events and increase or decrease Amazon Kinesis shards and updates alarms with new shards counter.

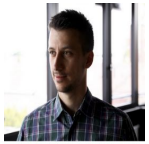


Contact Info

- **Arconsis**:
  - Website: https://www.arconsis.com
  - Github: https://github.com/arconsis

- **Dimos Botsaris**:
  - Website: https://www.eldimious.com/
  - Github: https://github.com/eldimious
  - Email: botsaris.d@gmail.com

- **Alexandros Koufatzis**:
  - Github: https://github.com/akoufa
  - Email: akoufa@gmail.com




Thank you
Alexandros and Dimosthenis
arconsis