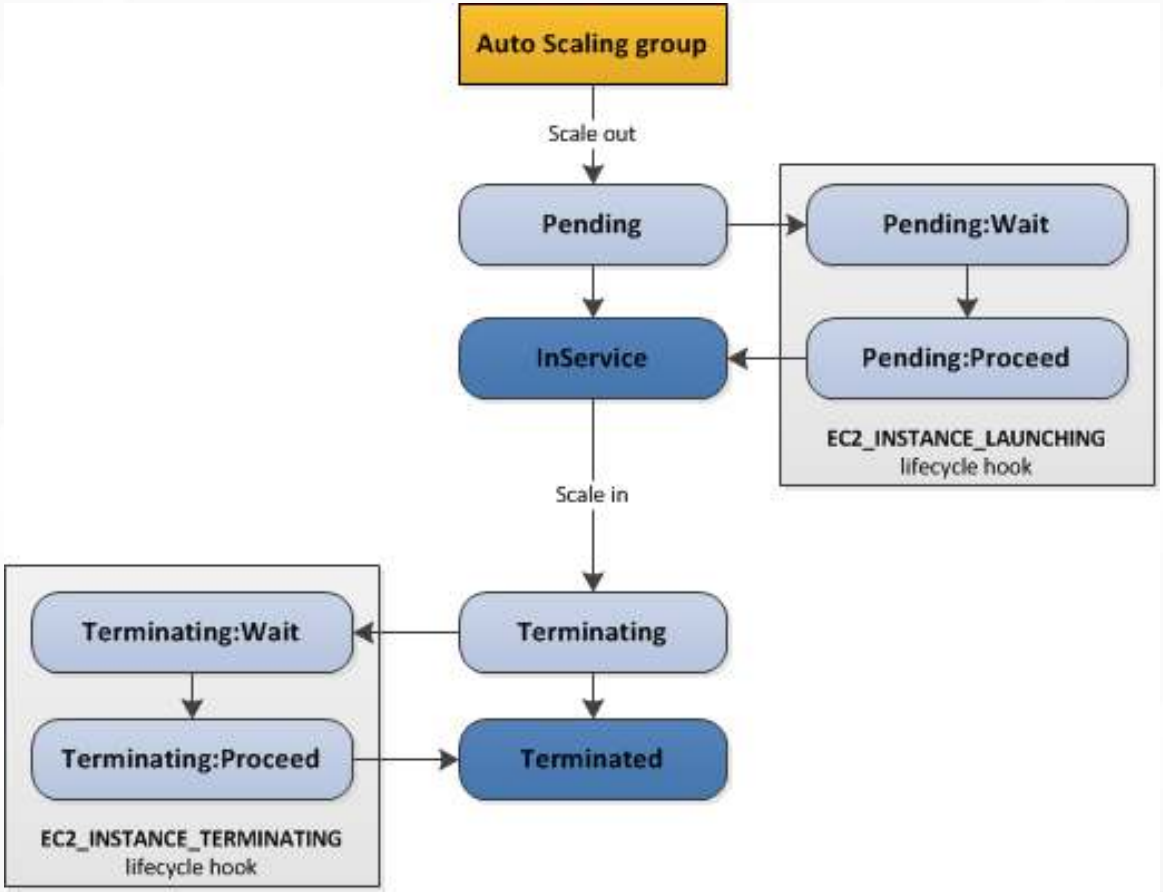
Using Amazon CloudWatch Events, AWS Lambda and Spark Streaming to Process EC2 Events

Julien Simon, Principal Technical Evangelist
julsimon@amazon.fr
@julsimon

27/04/2016

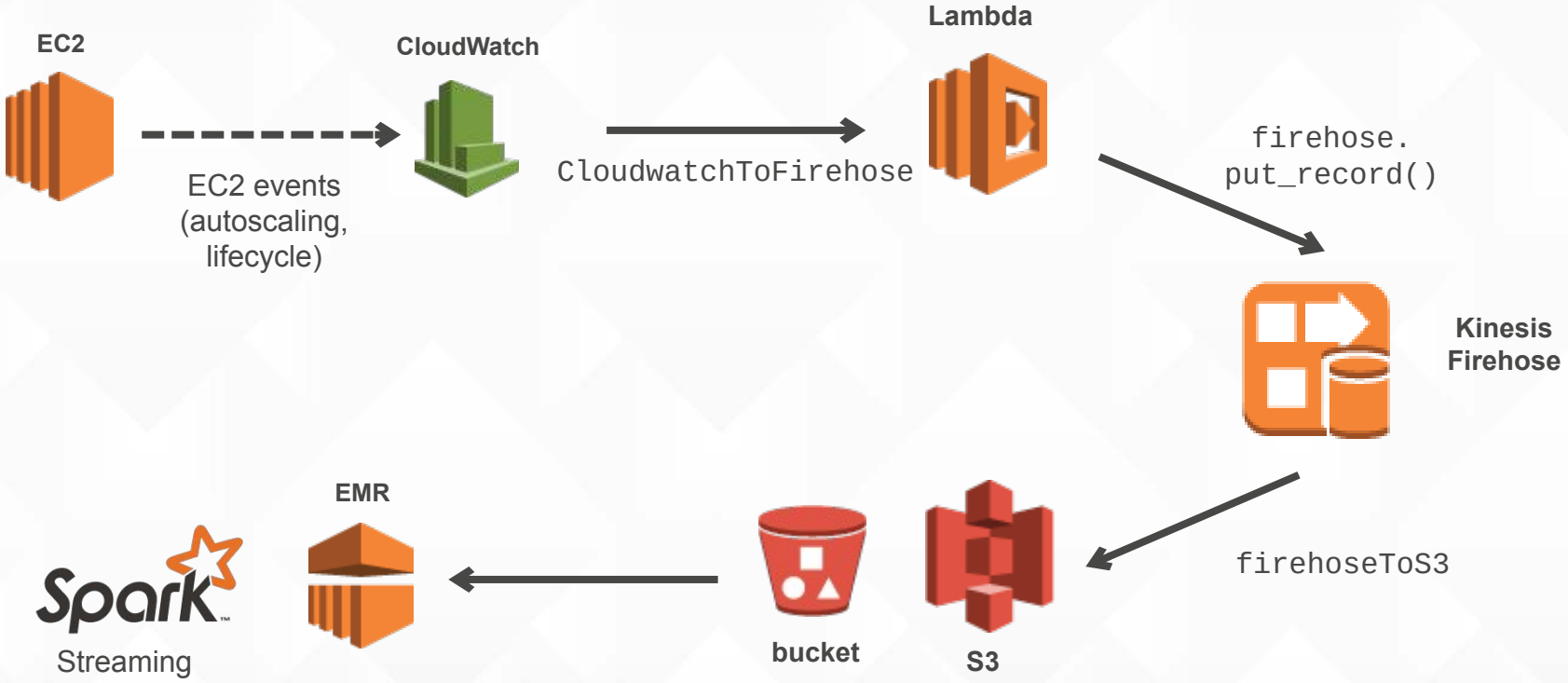
EC2 Instance Lifecycle



EC2 Lifecycle Hooks – what are they good for?

- Assign Elastic IP address on launch
- Register new instances with DNS, message queues,…
- Pull down log files before instance is terminated
- Investigate issues with an instance before terminating it
- Scaling containers on an ECS cluster

What we're going to build



EC2 event sources in CloudWatch Rules

Auto Scaling events

```
{
  "source": [
    "aws.autoscaling"
  ],
  "detail-type": [
    "EC2 Instance Launch Successful",
    "EC2 Instance Terminate Successful",
    "EC2 Instance Launch Unsuccessful",
    "EC2 Instance Terminate Unsuccessful",
    "EC2 Instance-launch Lifecycle Action",
    "EC2 Instance-terminate Lifecycle Action"
  ]
}
```

Lifecycle events

```
{
  "source": [
      "aws.ec2"
  ],
  "detail-type": [
      "EC2 Instance State-change Notification"
  ]
}
```

CLI: from CloudWatch Events to Lambda

```
% aws lambda get-function --function-name CloudwatchToFirehose \
  --query "Configuration.FunctionArn”

% aws events put-rule --name EC2AutoScaling \
  --event-pattern file://EC2AutoScaling.json --state ENABLED

% aws events put-targets --rule EC2AutoScaling \
  --targets Id=1,Arn=FUNCTION_ARN

% aws lambda add-permission --function-name CloudwatchToFirehose \
  --statement-id 1 --action 'lambda:InvokeFunction' \
  --principal events.amazonaws.com --source-arn RULE_ARN
```

Lambda function

```
def lambda_handler(event, context):
    print("Received EC2 event: " + json.dumps(event, indent=2))
    firehose = boto3.client('firehose')
    firehose.put_record(
        DeliveryStreamName="firehoseToS3”
        Record={"Data":json.dumps(event)}
    )
```

Reminder

a CloudWatch log group is created automatically for each version of your Lambda function

```
% aws logs describe-log-streams --log-group-name /aws/lambda/CloudwatchToFirehose \
  --query "logStreams[].logStreamName"
% aws logs get-log-events --log-stream-name LOG_STREAM_NAME \
  --log-group-name /aws/lambda/CloudwatchToFirehose
```

CLI: from Kinesis Firehose to S3

```
aws firehose create-delivery-stream \
--delivery-stream-name firehoseToS3 \
--s3-destination-configuration \
RoleARN=FIREHOSETOS3_ROLE_ARN, \
BucketARN="arn:aws:s3:::jsimon-public", \
Prefix="firehose", \
BufferingHints=\{SizeInMBs=1,IntervalInSeconds=60\}, \
CompressionFormat=”UNCOMPRESSED", \
EncryptionConfiguration={NoEncryptionConfig="NoEncryption"}
```

Spark Streaming

```
import org.apache.log4j.Logger
import org.apache.log4j.Level
import org.apache.spark.streaming.Seconds

Logger.getLogger("org").setLevel(Level.ERROR)
Logger.getLogger("com").setLevel(Level.ERROR)
Logger.getLogger("akka").setLevel(Level.ERROR)

val hadoopConf=sc.hadoopConfiguration;
hadoopConf.set("fs.s3.impl", "org.apache.hadoop.fs.s3native.NativeS3FileSystem")
hadoopConf.set("fs.s3.awsAccessKeyId”, ACCESS_KEY_ID)
hadoopConf.set("fs.s3.awsSecretAccessKey”, SECRET_ACCESS_KEY)
val ssc = new org.apache.spark.streaming.StreamingContext(sc,Seconds(10))
val lines = ssc.textFileStream("s3n://BUCKET_NAME")
lines.print()
ssc.start()
```