Serverless PHP

@tobiasnyholm

happyr

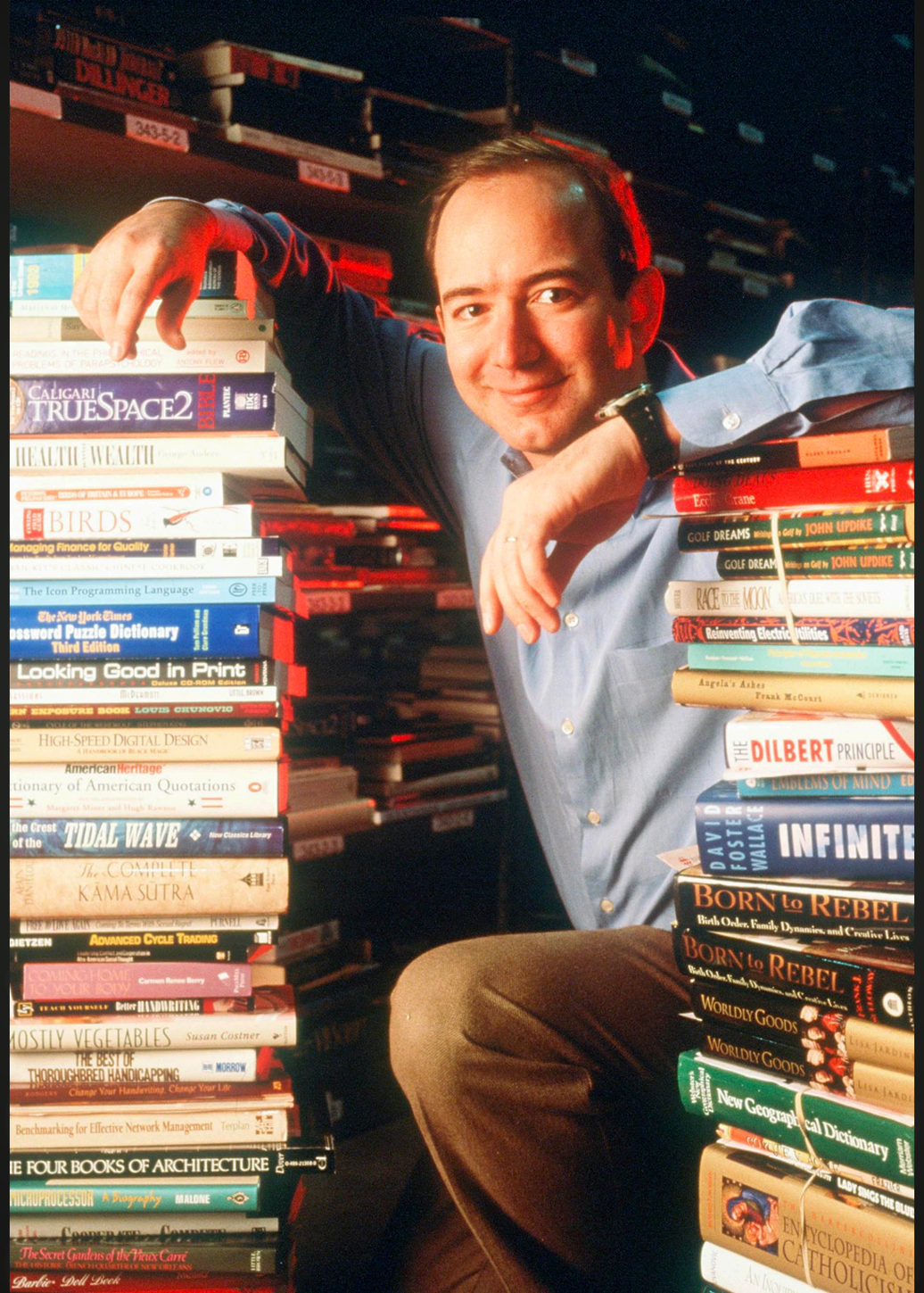
Serverless

Why?

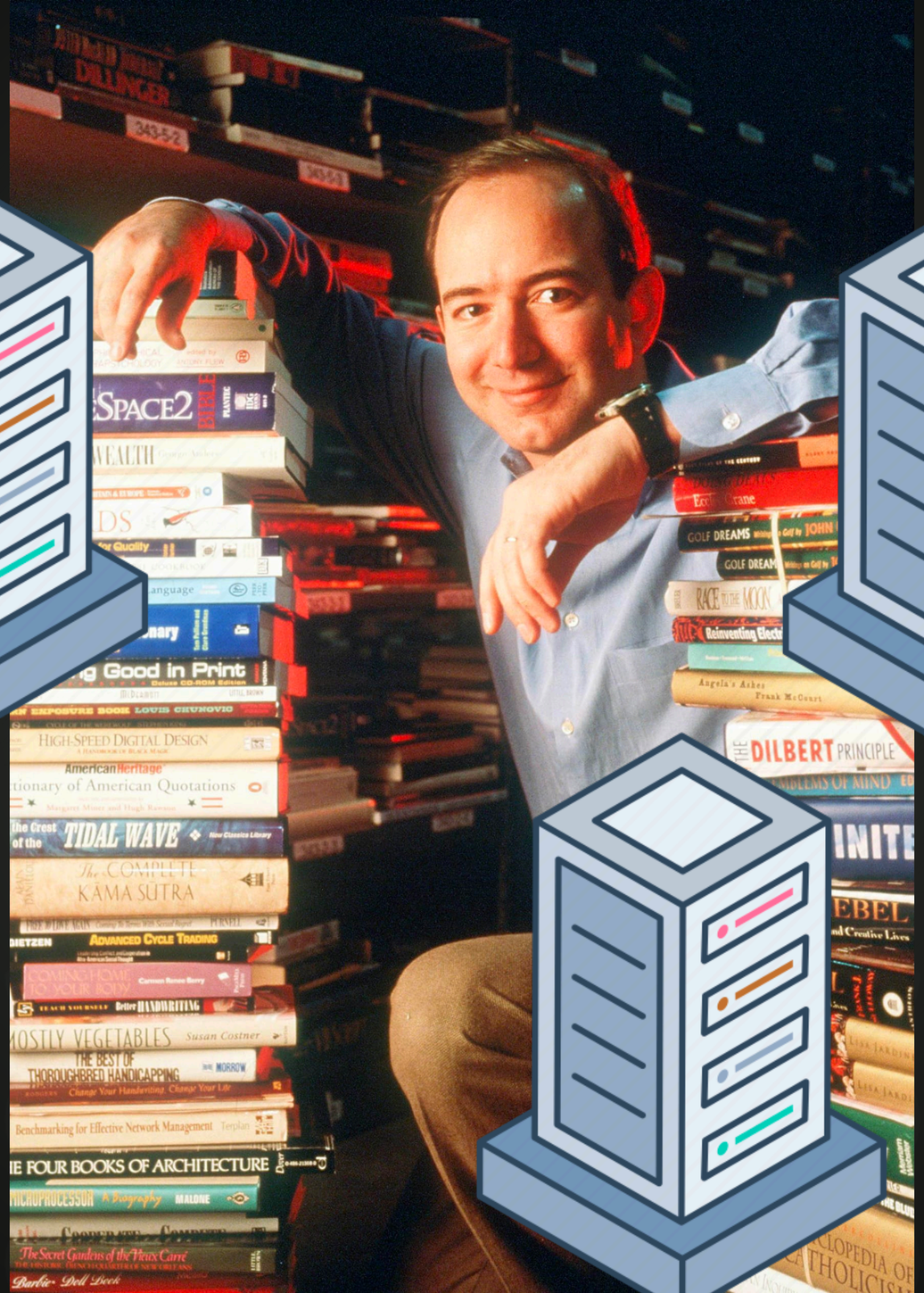
How?

There are still servers

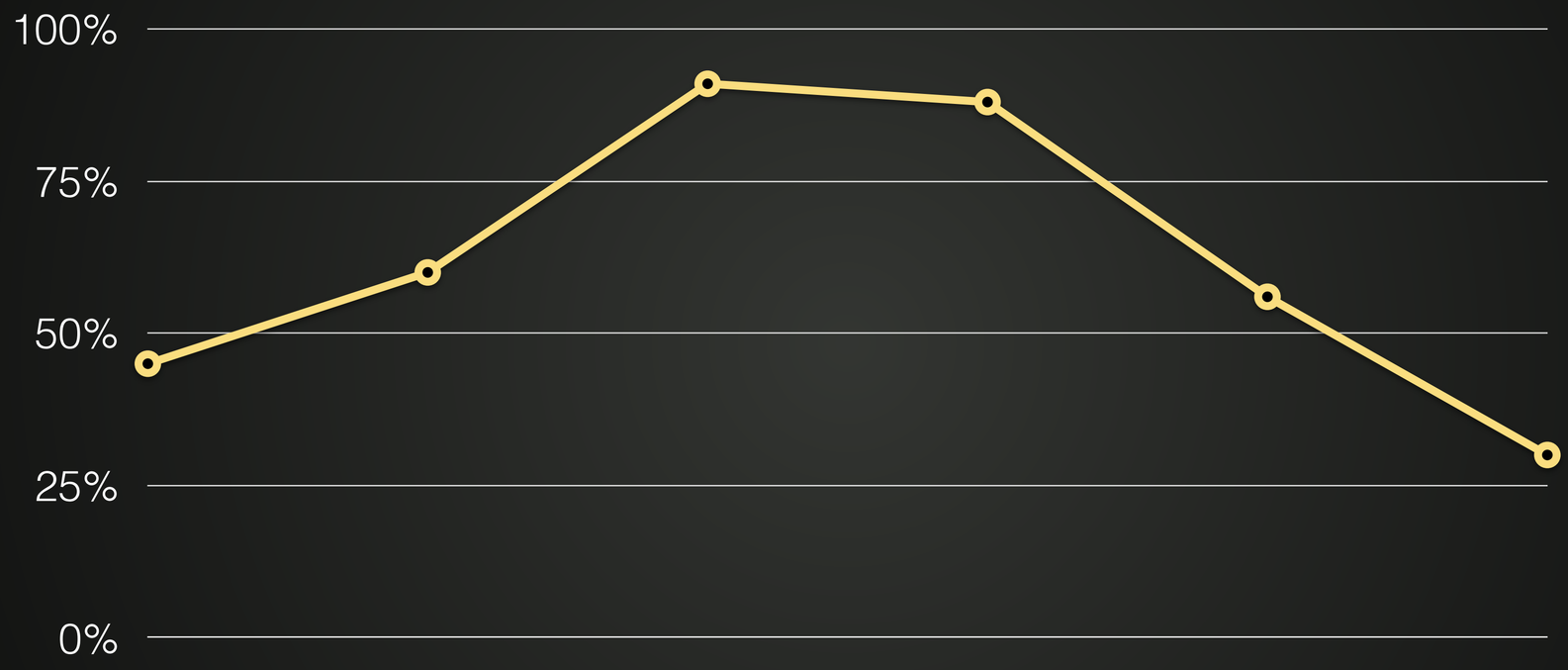
Fake History



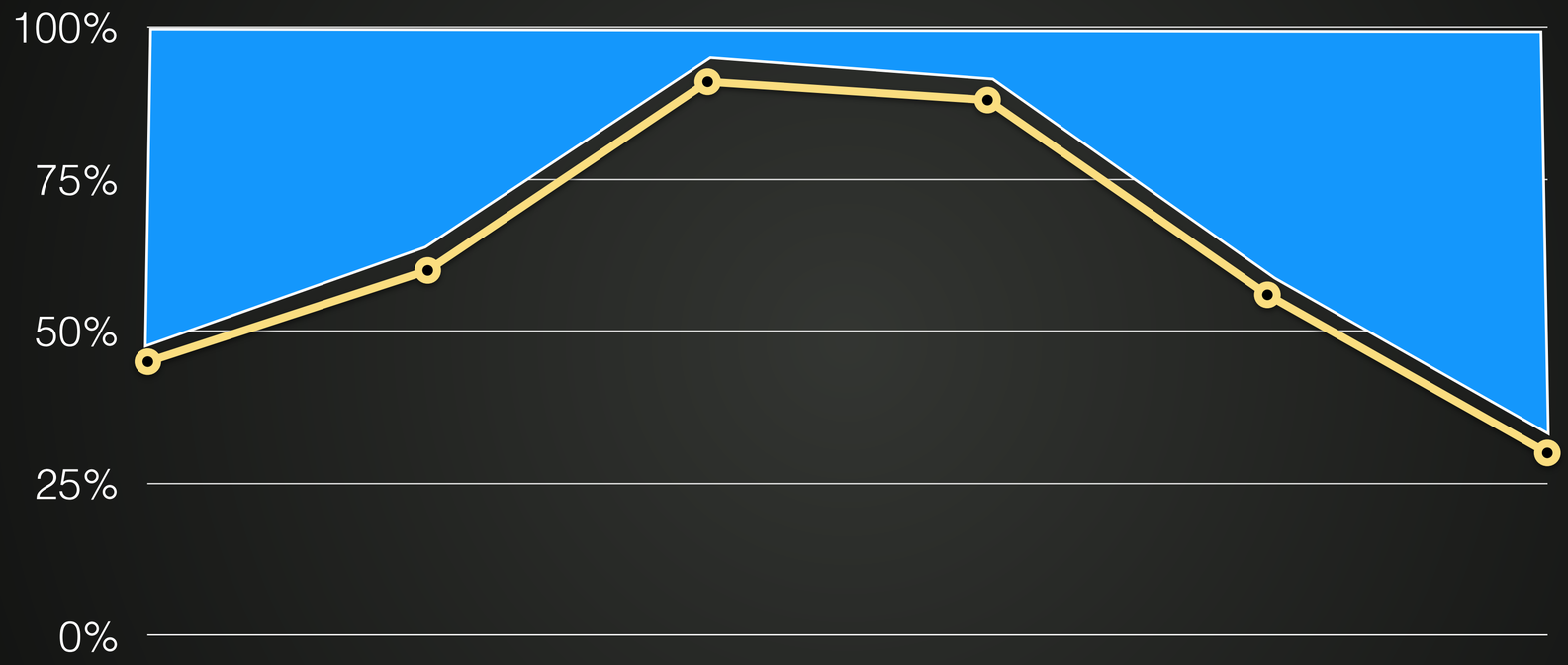
@tobiasnyholm



Machine usage



Machine usage



AWS Lambda

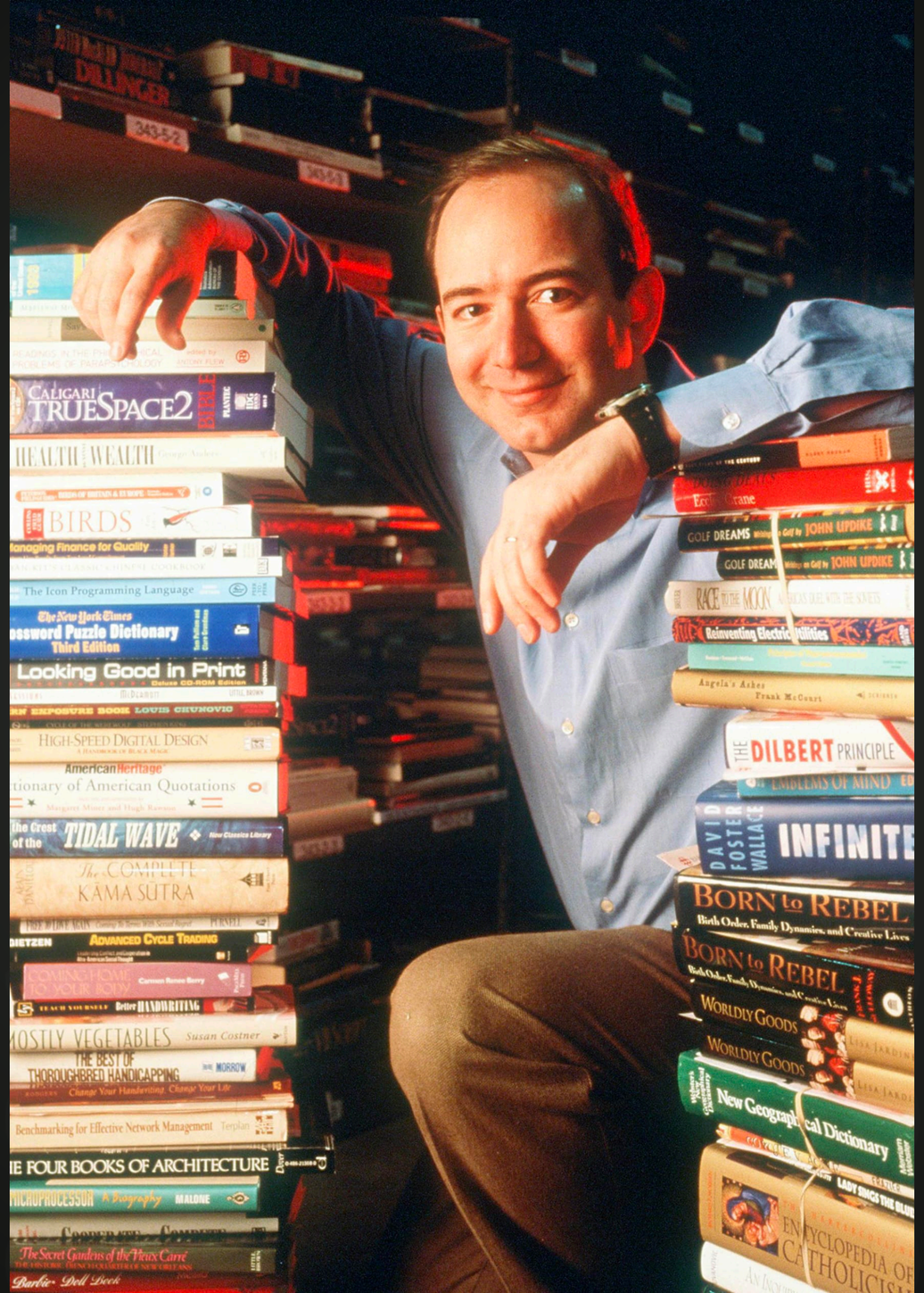
AWS Lambda

Fall 2018

```
function exec(cmd, handler)
{
    const childfork = require('child_process');

    return childfork.exec(cmd, handler);
}

exec("/usr/bin/php " + filename, myHandler);
```



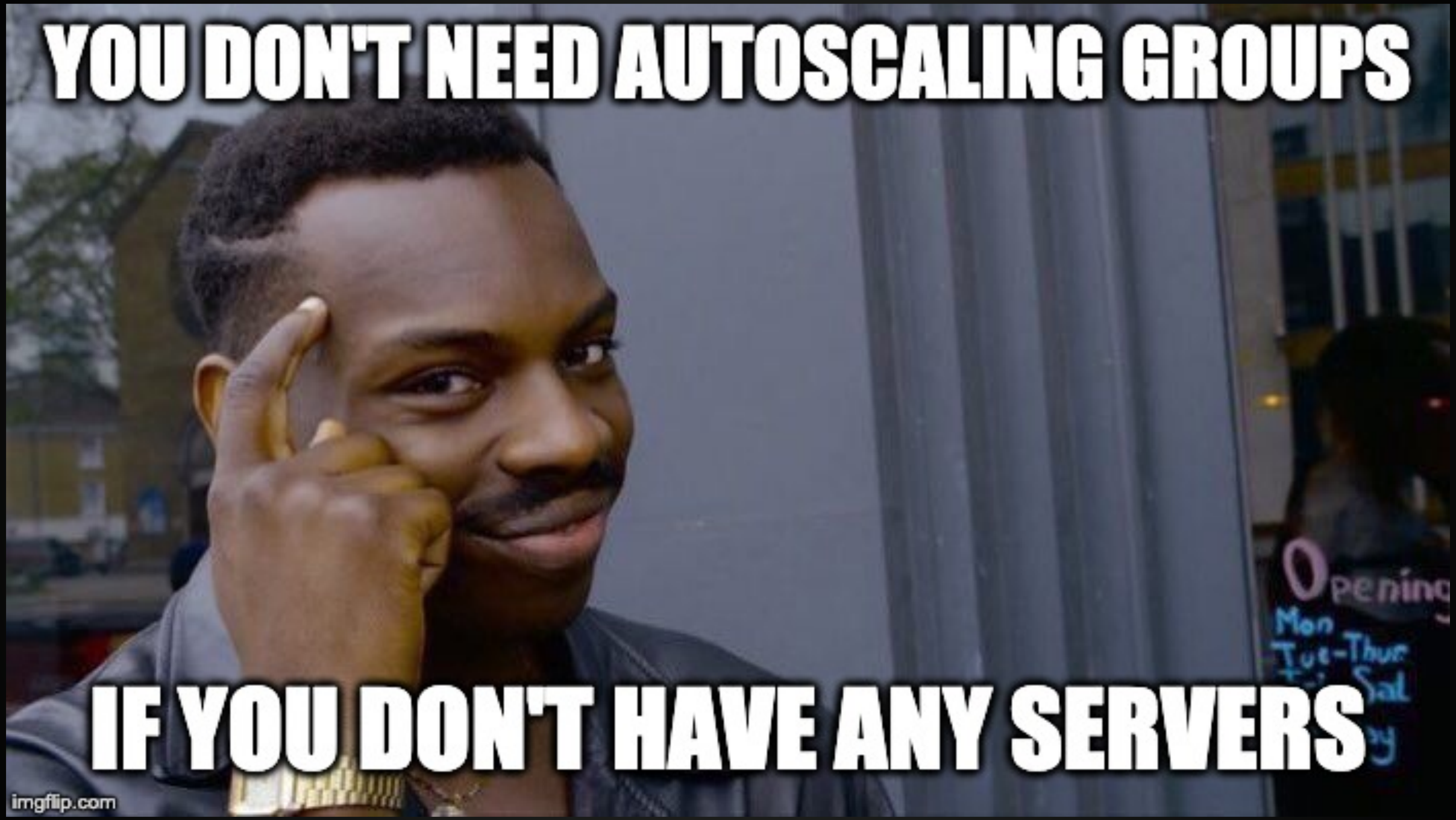
@tobiasnyholm

FROM amazonlinux:2018.03

Why?

3 killer reasons for Serverless

- Scales
- Cheap
- Easy to setup


YOU DON'T NEED AUTOSCALING GROUPS
IF YOU DON'T HAVE ANY SERVERS
imgflip.com

```php
<?php

function foo($x)
{
    return ($x >= 1) && ($x <= 1) && ($x !== 1);
}
```

foo(1.) Submit *Score: 2*

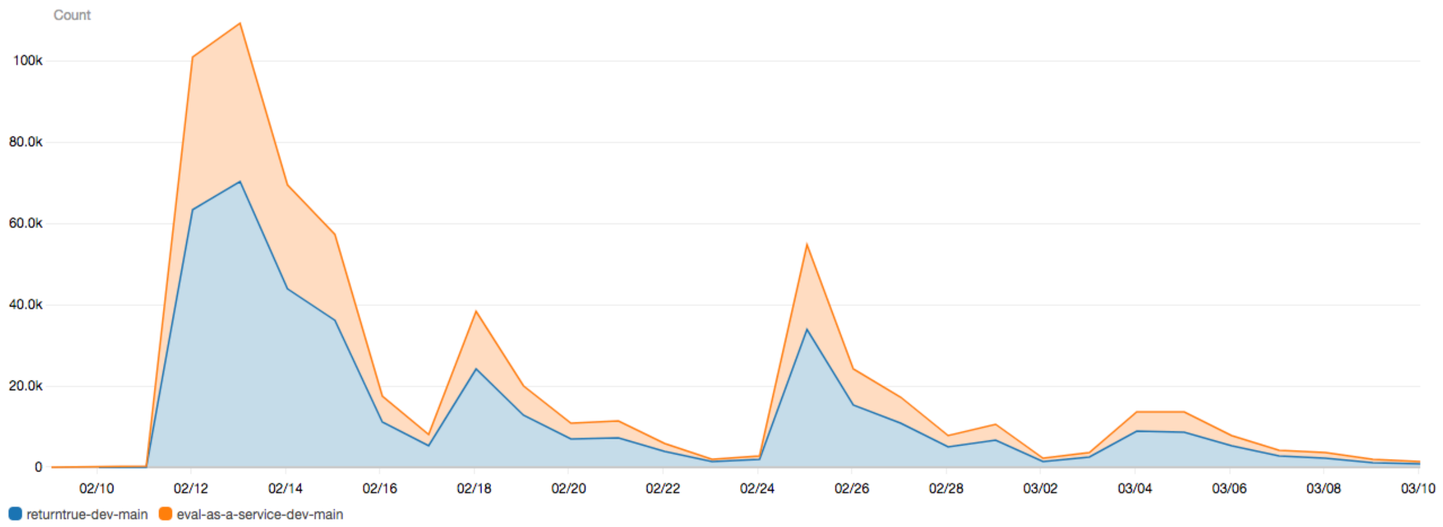
```
true
```

Well done! Move to the next step

Return true to win

returntrue.win

Matthieu Napoli



400k HTTP requests

650k executions

Matthieu Napoli

Show some code

$ composer req bref/bref

```php
require __DIR__./'vendor/autoload.php';

λ(function (array $event) {
    return 'hello ' . $event['name'];
});
```

$ composer req bref/bref

```php
require __DIR__./'vendor/autoload.php';

lambda(function (array $event) {
    return 'hello ' . $event['name'];
});
```

```
#!/bin/sh

# Fail on error
set -e

while true
do
    # We redirect stderr to stdout so that everything
    # written on the output ends up in Cloudwatch automatically
    /opt/bin/php $LAMBDA_TASK_ROOT/$_HANDLER 2>&1
done
```

Serverless is easy

```yaml
service: app
provider:
    name: aws
    runtime: provided
plugins:
    - ./vendor/bref/bref
functions:
    website:                          ••
        handler: index.php
        layers:
            - ${bref:layer.php-73-fpm}
        # Configuration of API Gateway
        events:
            -   http: 'ANY /'
            -   http: 'ANY /{proxy+}'
```

```php
echo 'hello world';
```

```php
#!/opt/bin/php
<?php declare(strict_types=1);

use Bref\Runtime\LambdaRuntime;
use Bref\Runtime\PhpFpm;

$appRoot = getenv('LAMBDA_TASK_ROOT');
require $appRoot . '/vendor/autoload.php';

$lambdaRuntime = LambdaRuntime::fromEnvironmentVariable();
$handler = $appRoot . '/' . getenv('_HANDLER');

$phpFpm = new PhpFpm($handler);
$phpFpm->start();

while (true) {
    $lambdaRuntime->processNextEvent(function ($event) use ($phpFpm): array {
        $response = $phpFpm->proxy($event);

        $multiHeader = array_key_exists('multiValueHeaders', $event);
        return $response->toApiGatewayFormat($multiHeader);
    });

    try {
        $phpFpm->ensureStillRunning();
    } catch (\Throwable $e) {
        echo $e->getMessage();
        exit(1);
    }
```

```php
use Bref\Runtime\LambdaRuntime;
use Bref\Runtime\PhpFpm;

$appRoot = getenv('LAMBDA_TASK_ROOT');
require $appRoot . '/vendor/autoload.php';

$lambdaRuntime = LambdaRuntime::fromEnvironmentVariable();
$handler = $appRoot . '/' . getenv('_HANDLER');

$phpFpm = new PhpFpm($handler);
$phpFpm->start();

while (true) {
    $lambdaRuntime->processNextEvent(function ($event) use ($phpFpm): array {
        $response = $phpFpm->proxy($event);

        $multiHeader = array_key_exists('multiValueHeaders', $event);
        return $response->toApiGatewayFormat($multiHeader);
    });

    try {
        $phpFpm->ensureStillRunning();
    } catch (\Throwable $e) {
        echo $e->getMessage();
        exit(1);
    }
}
```

Serverless is easy

Only /tmp is writeable

```php
class Kernel
{
    // ...
    public function getLogDir()
    {
        // When on the lambda only /tmp is writeable
        if (getenv('LAMBDA_TASK_ROOT') !== false) {
            return '/tmp/log/';
        }

        return $this->getProjectDir().'/var/log';
    }

    public function getCacheDir()
    {
        // When on the lambda only /tmp is writeable
        if (getenv('LAMBDA_TASK_ROOT') !== false) {
            return '/tmp/cache/'.$this->environment;
        }

        return $this->getProjectDir().'/var/cache/'.$this->environment;
    }
}
```

Bref with Laravel

https://bref.sh/docs/frameworks/laravel.html

Cold starts

```yaml
service: app
provider:
    name: aws
    runtime: provided
plugins:
    - ./vendor/bref/bref
functions:
    website:
        handler: index.php
        layers:
            - ${bref:layer.php-73-fpm}
        # Configuration of API Gateway
        events:
            - http: 'ANY /'
            - http: 'ANY /{proxy+}
            - schedule:
                rate: rate(5 minutes)
                input:
                    warmer: true
```

Cron

```php
require __DIR__./'vendor/autoload.php';

lamba(function (array $event) {
    return 'hello ' . $event['name'];
});
```

```yaml
events:
    - schedule:
        schedule: cron(0/1 * * * ? *)
        input:
            foo: bar
```

PHP extensions

Extensions installed and enabled

- Core
- bcmath
- ctype
- curl
- date
- dom
- exif
- fileinfo
- filter
- ftp
- gettext
- hash
- iconv
- json
- libxml
- mbstring
- mysqli
- mysqlnd
- opcache
- openssl
- pcntl
- pcre
- PDO
- pdo_sqlite
- Phar
- posix
- readline
- Reflection
- session
- SimpleXML
- sodium
- SOAP
- sockets
- SPL
- sqlite3
- standard
- tokenizer
- xml
- xmlreader
- xmlwriter
- zlib

Extensions installed but disabled by default

- **intl** - Internationalization extension (referred as Intl) is a wrapper for ICU library, enabling PHP programmers to perform various locale-aware operations.
- **APCu** - APCu is APC stripped of opcode caching.
- **phpredis** - The phpredis extension provides an API for communicating with the Redis key-value store.
- **PostgreSQL PDO Driver** - PDO_PGSQL is a driver that implements the PHP Data Objects (PDO) interface to enable access from PHP to PostgreSQL databases.
- **MySQL PDO Driver** - PDO_MYSQL is a driver that implements the PHP Data Objects (PDO) interface to enable access from PHP to MySQL databases.
- **Mongodb** - Unlike the mongo extension, this extension is developed atop the » libmongoc and » libbson libraries. It provides a minimal API for core driver functionality: commands, queries, writes, connection management, and BSON serialization.
- **pthreads** - pthreads is an object-orientated API that provides all of the tools needed for multi-threading in PHP. PHP applications can create, read, write, execute and synchronize with Threads, Workers and Threaded objects.


My App


My App


My App


My App

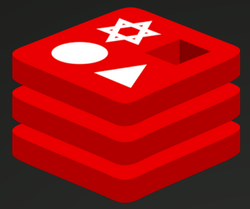
PHP extensions

http://developer.happyr.com/newrelic-on-bref-aws-lambda

happyr

Architecture

Front door

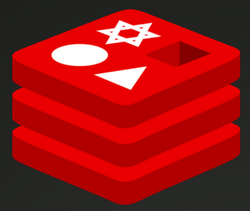


Analytics

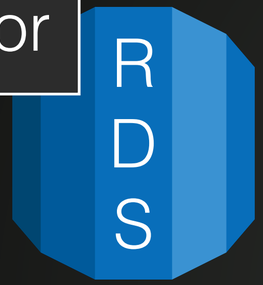
Match Engine

Measure

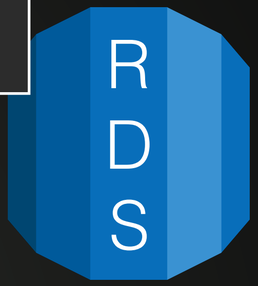
happyr



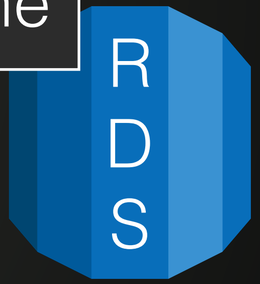
Front door



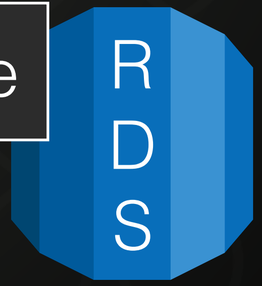
Analytics



Match Engine

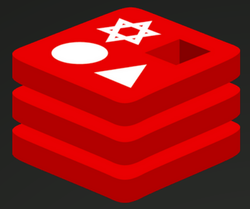


Measure



happyr

Front door

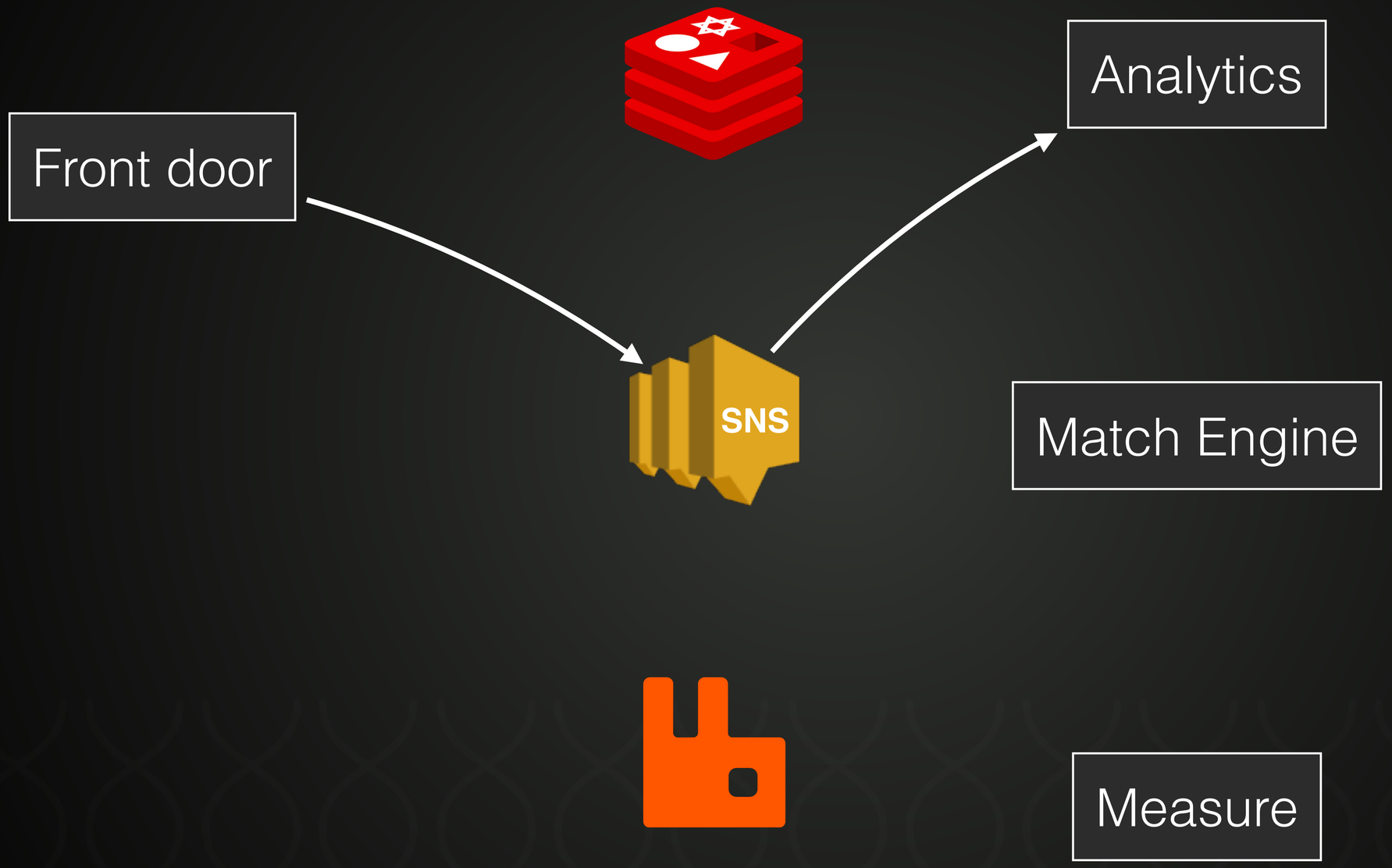


Analytics

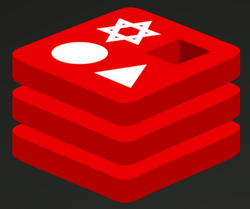
Match Engine

Measure

happyr


Front door
SNS
Analytics
Match Engine
Measure

Front door

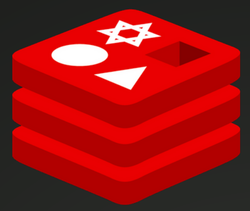


Analytics

Match Engine

Measure

happyr



Analytics

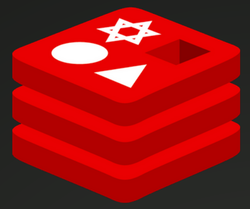
Front door

Match Engine

Measure

happyr

Front door

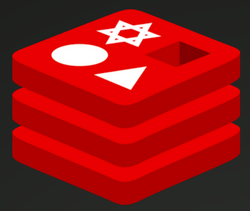


Analytics

Match Engine

Measure

happyr



Analytics

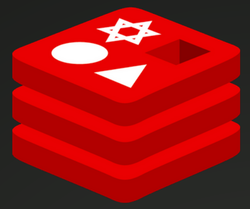
Front door

Match Engine

Measure

happyr

Front door

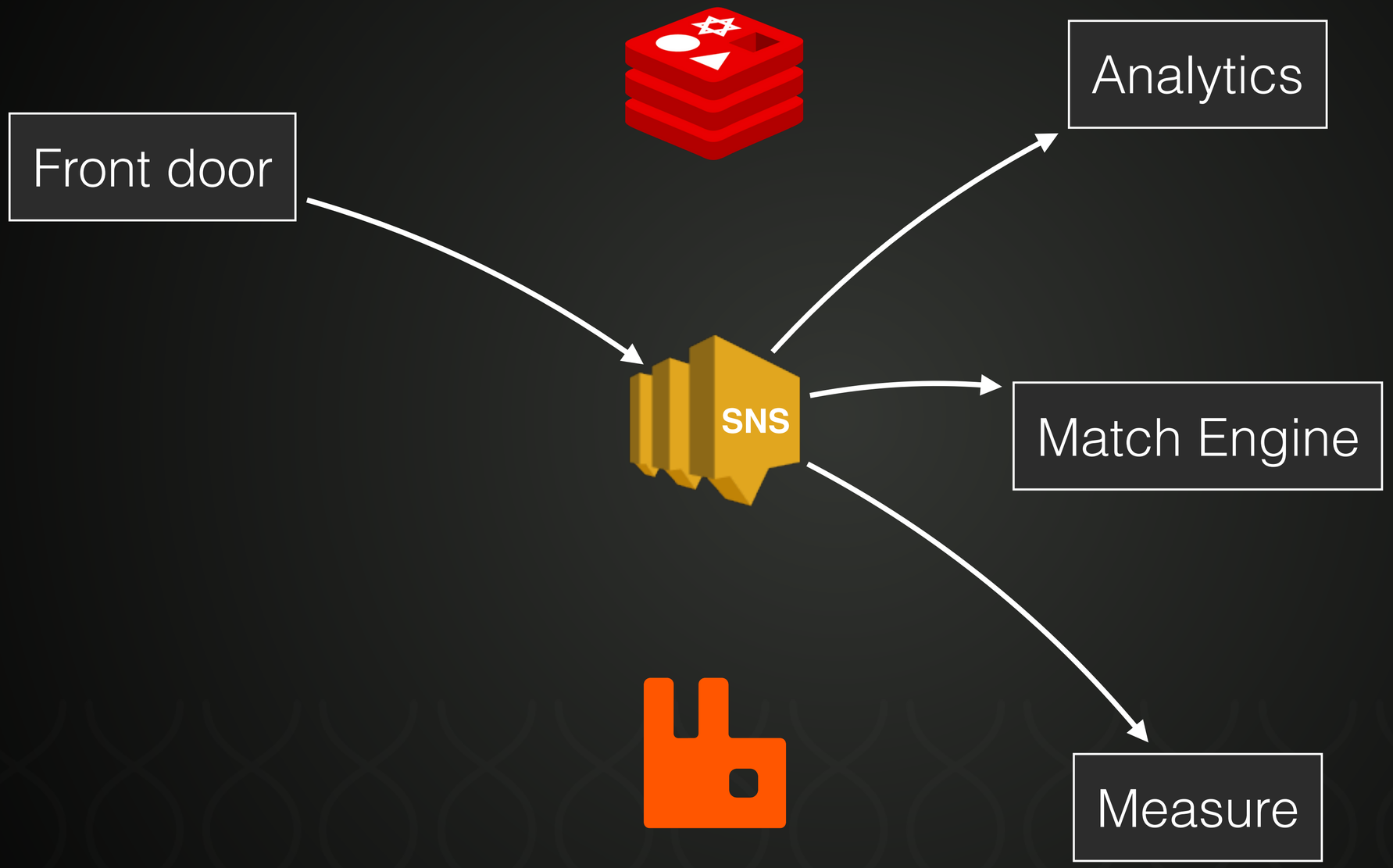


Analytics

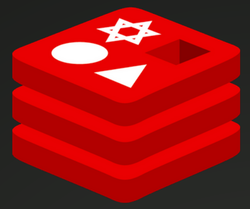
Match Engine

Measure

happyr


Front door
SNS
Analytics
Match Engine
Measure

Front door

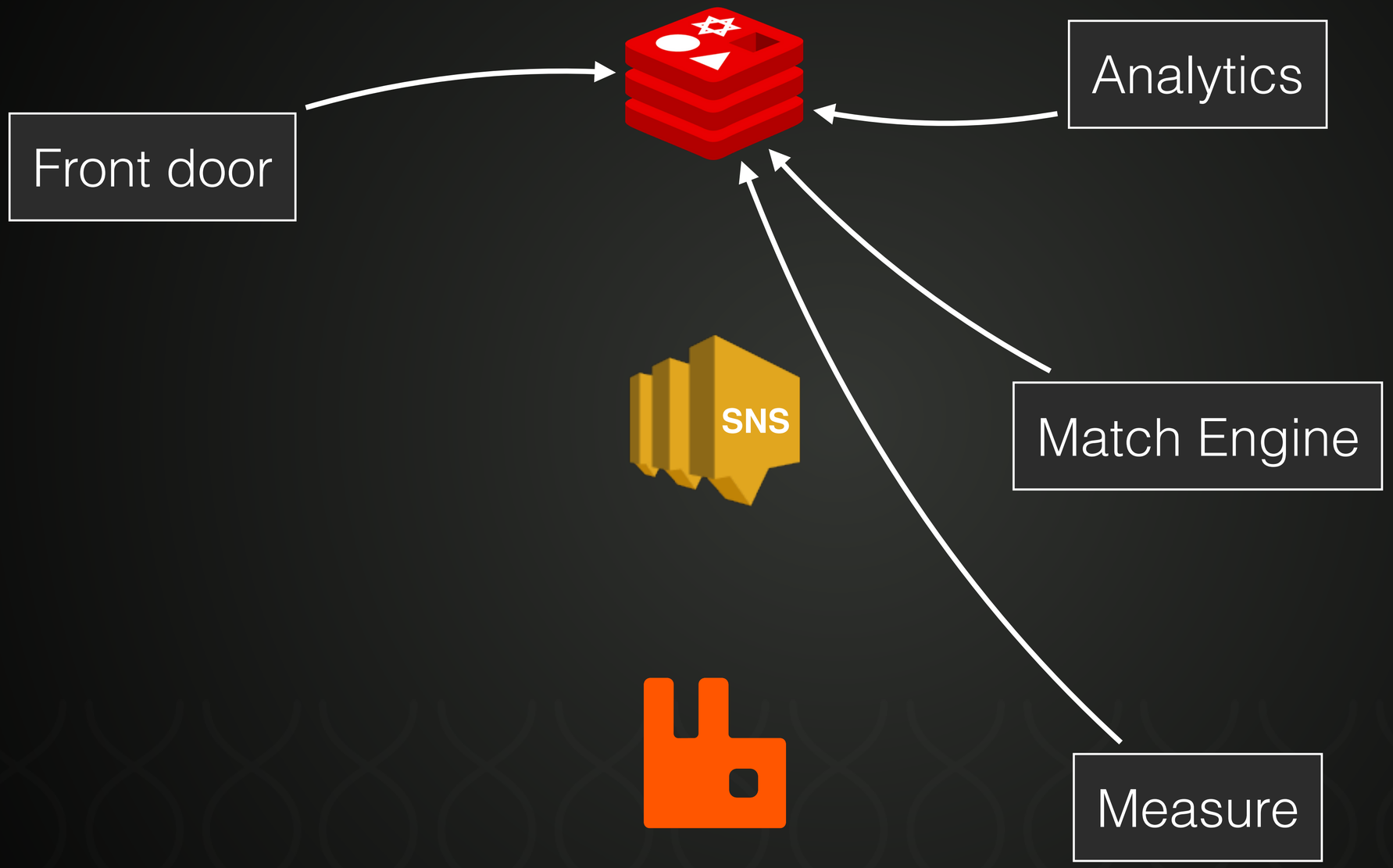


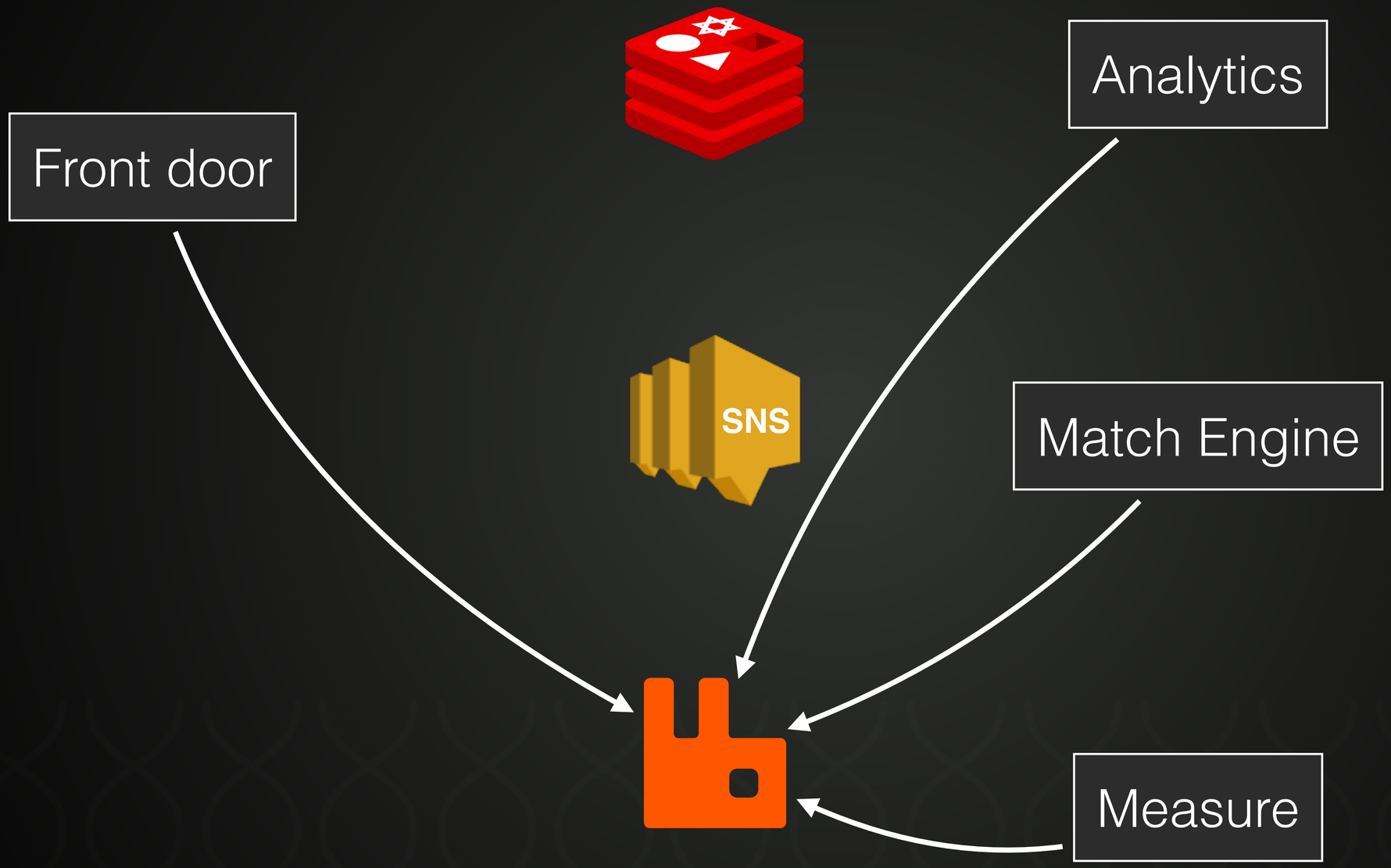
Analytics

Match Engine

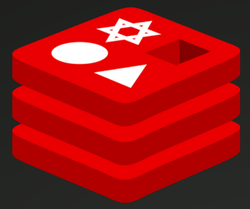
Measure

happyr


Front door
Analytics
SNS
Match Engine
Measure


Front door
Analytics
SNS
Match Engine
Measure

Front door



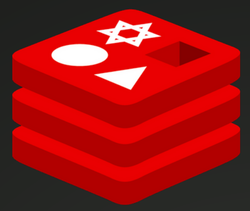
Analytics

Match Engine

Retry app

Measure

happyr



Analytics

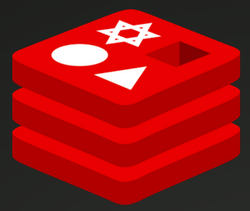
Front door

Match Engine

Retry app

Measure

happyr



Analytics

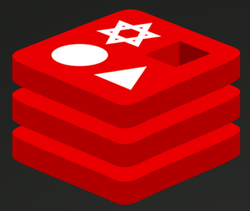
Front door

Match Engine

Retry app

Measure

happyr



Analytics

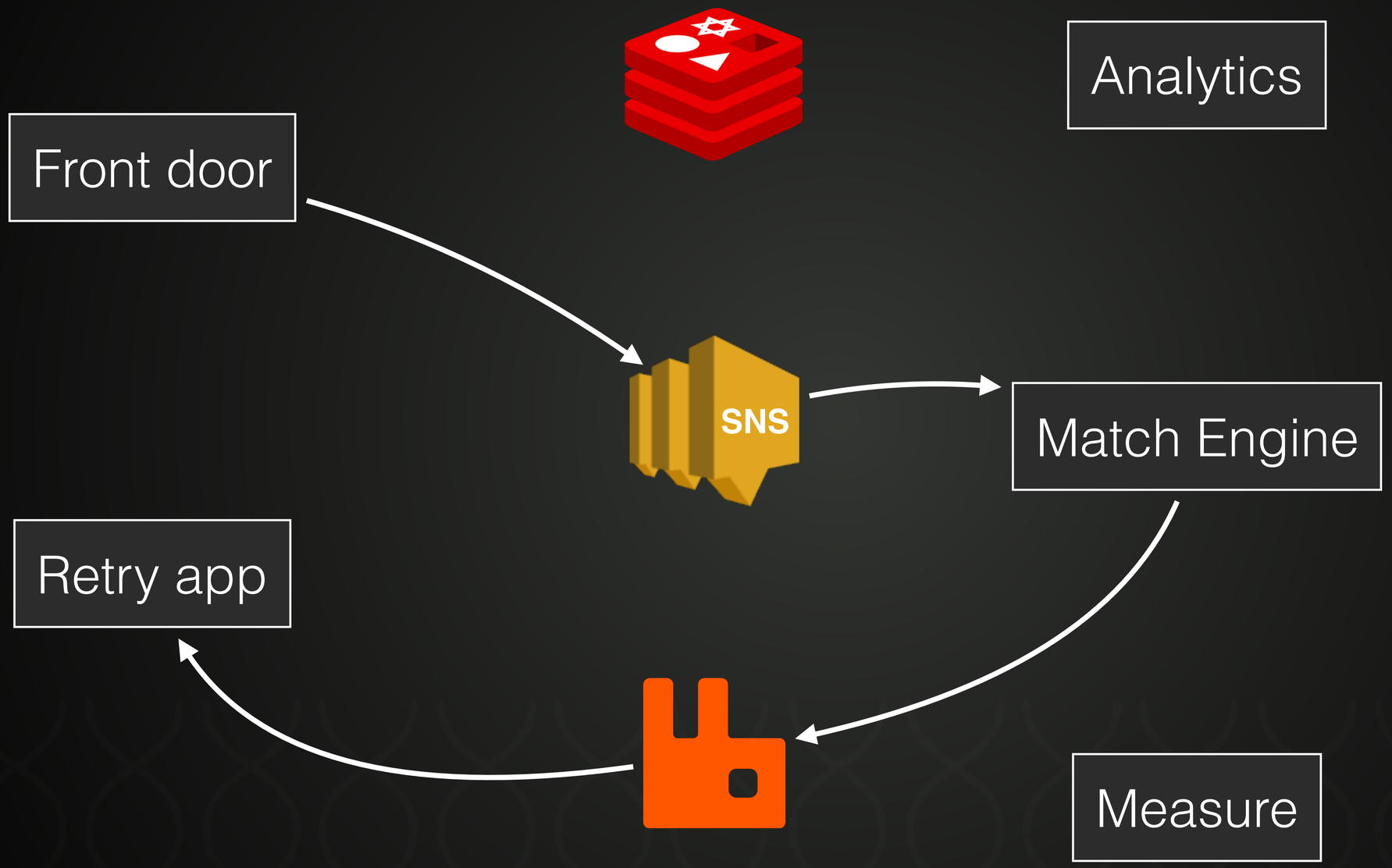
Front door

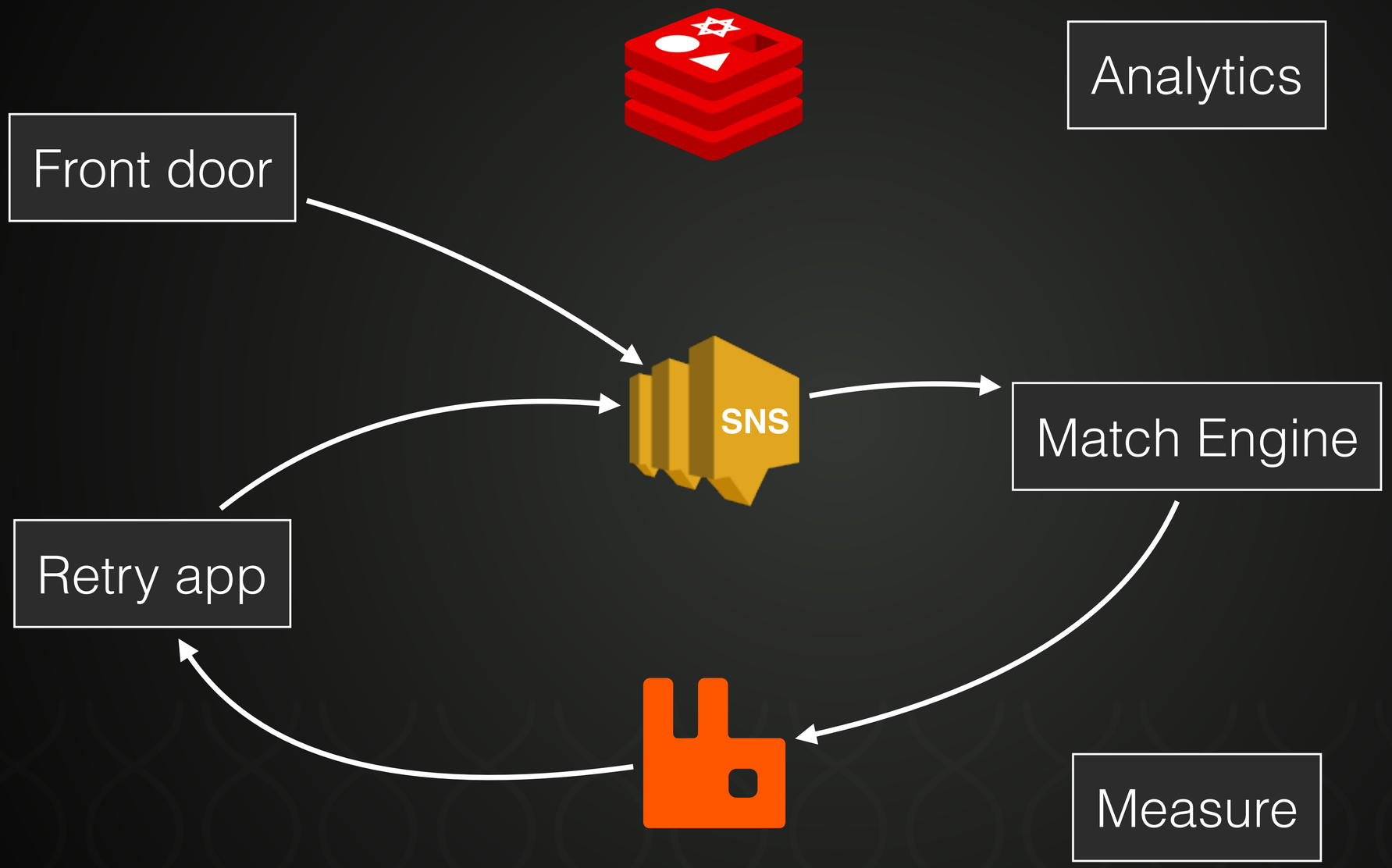
Match Engine

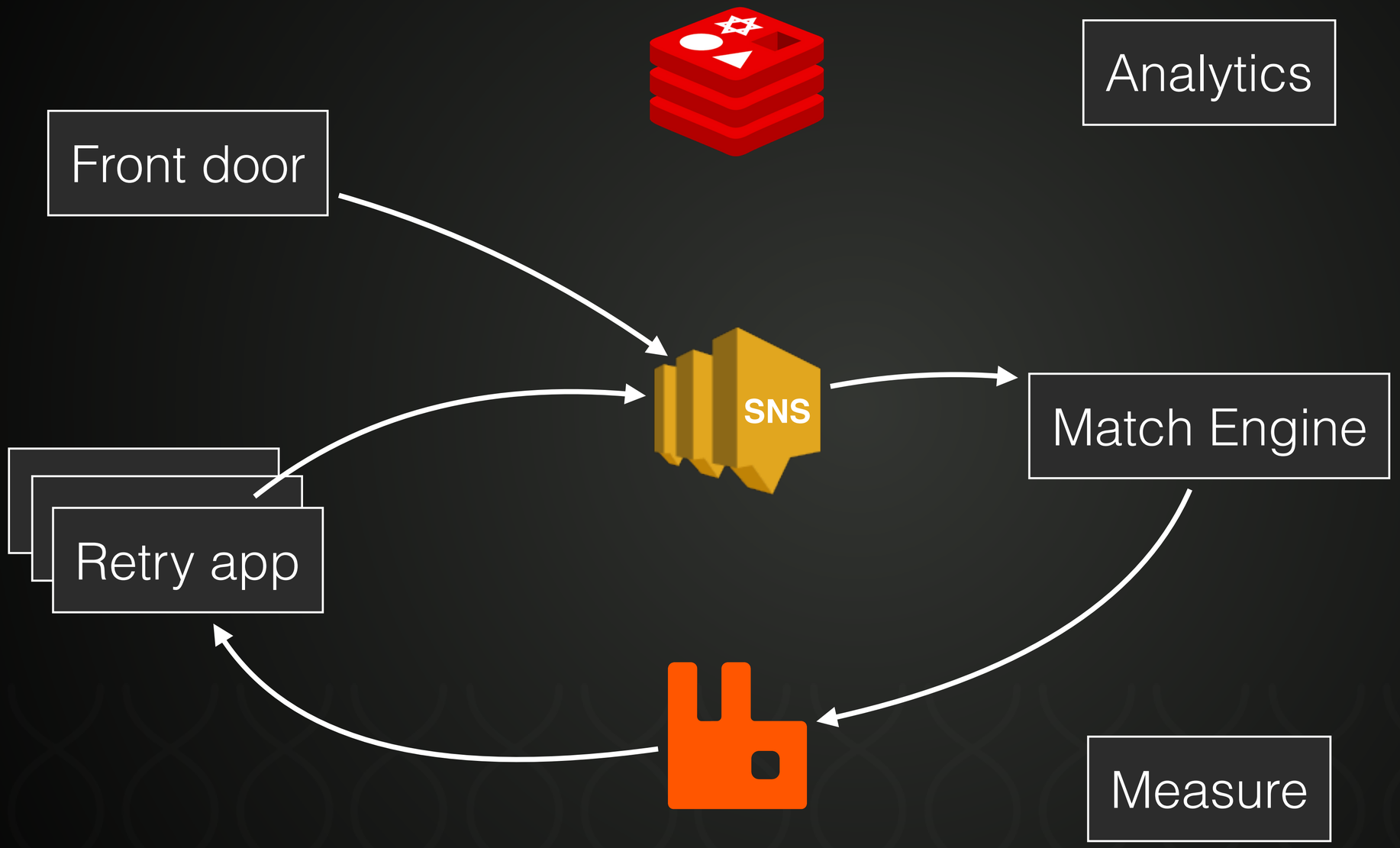
Retry app

Measure

happyr


Analytics
Front door
SNS
Match Engine
Retry app
Measure


Analytics
Front door
SNS
Match Engine
Retry app
Measure


Analytics
Front door
SNS
Match Engine
Retry app
Measure

Messaging

http://developer.happyr.com/symfony-messenger-on-aws-lambda

@tobiasnyholm

happyr

```php
include \dirname(__DIR__).'/vendor/autoload.php';

$publisher = new \Aws\Sns\SnsClient([
    'version' => '2010-03-31',
    'region' => \getenv('AWS_TARGET_REGION'),
]);

$publisher->publish([
    'TopicArn' => 'my_topic_arn',
    'Message' => 'my_json_message',
]);
```

```php
use App\Kernel;
use Rawls\SnsConsumer\SnsConsumer;


lambda(static function (array $event) {
    $kernel = new Kernel($_SERVER['APP_ENV'], (bool) $_SERVER['APP_DEBUG']);
    $kernel->boot();
    $container = $kernel->getContainer();

    /** @var SnsConsumer $consumer */
    $consumer = $container->get(SnsConsumer::class);

    foreach ($event['Records'] as $record) {
        if (!isset($record['Sns'])) {
            continue;
        }

        $consumer->consume($record['Sns']);
    }

    return 'OK.';
});
```

```php
final class SnsConsumer
{
    public function consume(array $event)
    {
        try {
            $envelope = $this->serializer->decode(['body' => $event['Message']]);
            $this->bus->dispatch($envelope);
        } catch (\Throwable $e) {
            if ($e instanceof MessageDecodingFailedException) {
                $prev = $e->getPrevious();
                if ($prev instanceof HydratorNotFoundException) {
                    // Dont retry.
                    return;
                }
            }

            $retryAttempt = (int) ($event['MessageAttributes']['retry_attempt']['Value'] ?? 0);

            $mqMessage = new Message($event['Message'], [
                'delivery_mode' => 2, // Persistent
                'content_type' => 'application/json',
                'headers' => [
                    'SnsTopicArn' => $event['TopicArn'],
                    'retry_attempt' => $retryAttempt + 1,
                ],
            ]);
            $this->channel->publish($mqMessage, 'sns_retry', $this->getRoutingKeyFromRetryAttempt($
        }
    }
```

@tobiasnyholm

```
            $retryAttempt = (int) ($event['MessageAttributes']['retry_attempt']['Value'] ?? 0);

            $mqMessage = new Message($event['Message'], [
                'delivery_mode' => 2, // Persistent
                'content_type' => 'application/json',
                'headers' => [
                    'SnsTopicArn' => $event['TopicArn'],
                    'retry_attempt' => $retryAttempt + 1,
                ],
            ]);
            $this->channel->publish($mqMessage, 'sns_retry', $this->getRoutingKeyFromRetryAttempt($
        }
    }

    private function getRoutingKeyFromRetryAttempt(int $retries): string
    {
        switch ($retries) {
            case 0:
            case 1:
                return 'retry_0';
            case 2:
            case 3:
                return 'retry_1';
            default:
                return 'retry_2';
        }
    }
}
```

Handle failure

http://developer.happyr.com/sns-retry

@tobiasnyholm

happyr

```php
use Rawls\QueueAdapter\ChannelFactory;
use Rawls\QueueAdapter\Impl\AmqpExtChannel;

include \dirname(__DIR__).'/vendor/autoload.php';

$channel = ChannelFactory::create(\getenv('HAPPYR_MQ'));
$publisher = new \Aws\Sns\SnsClient([
    'version' => '2010-03-31',
    'region' => \getenv('AWS_TARGET_REGION'),
]);

lambda(function (array $event) use ($channel, $publisher) {
    if (!isset($event['queue'])) {
        throw new RuntimeException('Queue must be defined');
    }

    echo \sprintf('Reading from queue: %s... ', $event['queue']); // No line break
    $count = 0;
    while ($message = $channel->get(false, $event['queue'])) {
        $attr = $message->getAttributes();
        if (!isset($attr['headers']['SnsTopicArn'])) {
            echo "No SnsTopicArn found \n";

            continue;
        }

        try {
            $publisher->publish([
```

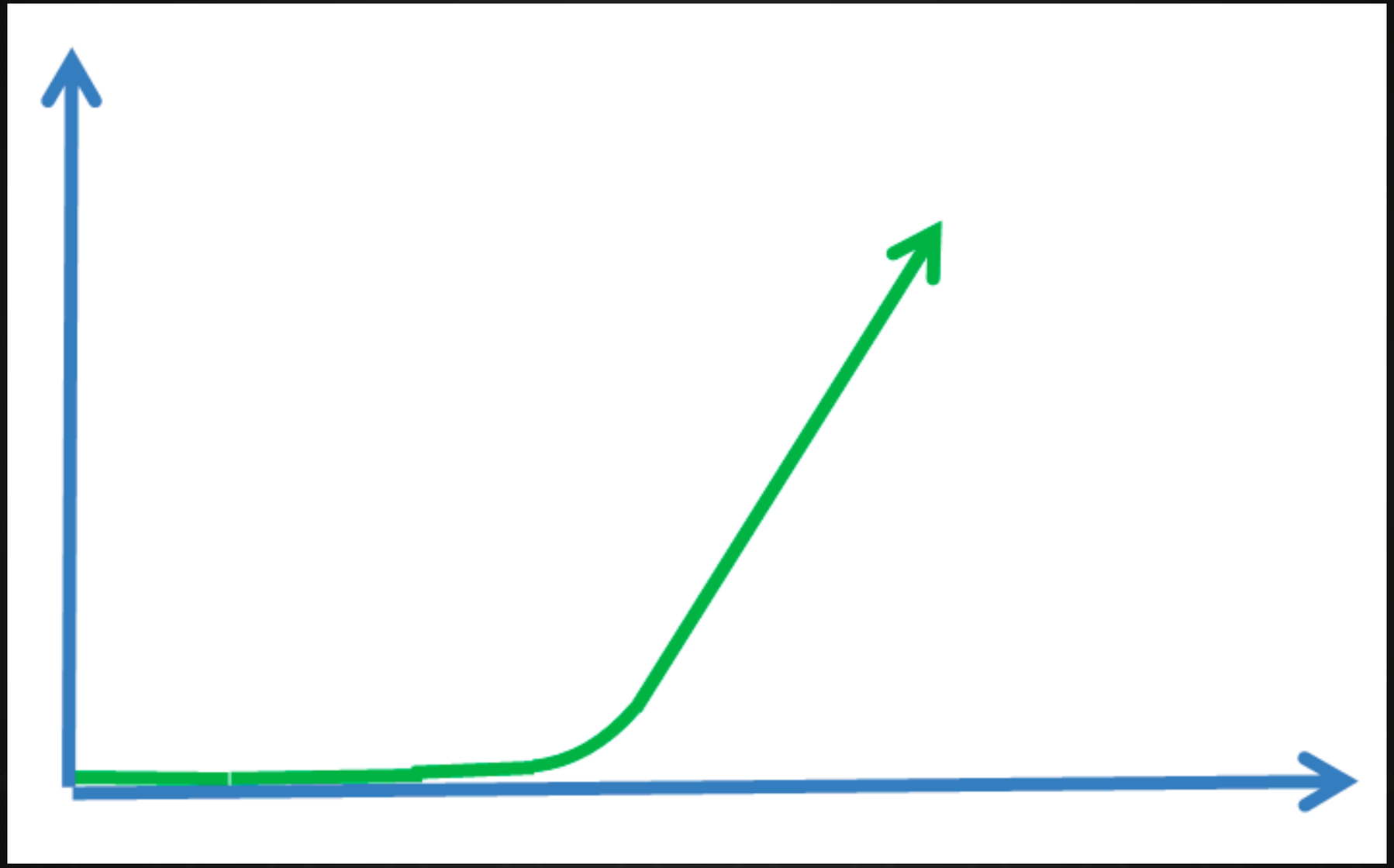
```
        try {
            $publisher->publish([
              'TopicArn' => $attr['headers']['SnsTopicArn'],
              'Message' => $message->getMessage(),
              'MessageAttributes' => [
                'application_id' => ['StringValue' => ''.$attr['headers']['application_id'] ?? ''],
                'retry_attempt' => ['StringValue' => ''.$attr['headers']['retry_attempt'] ?? 1],
              ],
            ]);
        } catch (\Throwable $throwable) {
            echo "\nCould not publish message \n";

            // Start handling other messages.
            continue;
        }

        // If we cannot publish the message, we will never ack it.
        $message->ack();
        ++$count;
    }

    if ($count > 0) {
        echo \sprintf('Republished %d messages.', $count)."\n";
    }
});
```

Next: QA environment



Future: Serverless Aurora Postgres

Questions?

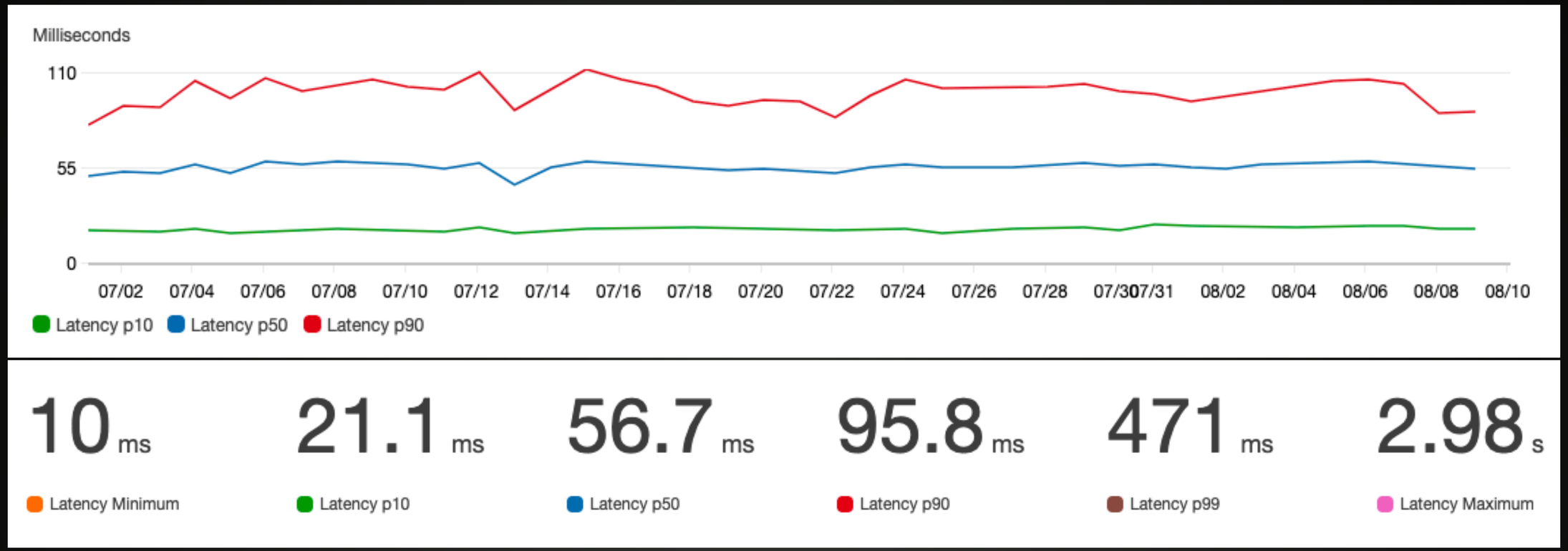
@tobiasnyholm

Costs?

https://cost-calculator.bref.sh/

Questions?

@tobiasnyholm



externals.io

Questions?

@tobiasnyholm

https:// bref.sh

https:// nyholm.tech