Purify your Lambdas

Luis Ángel Vicente Sánchez

Purify your Amazon Lambdas

Luis Ángel Vicente Sánchez

What is Amazon Lambda?

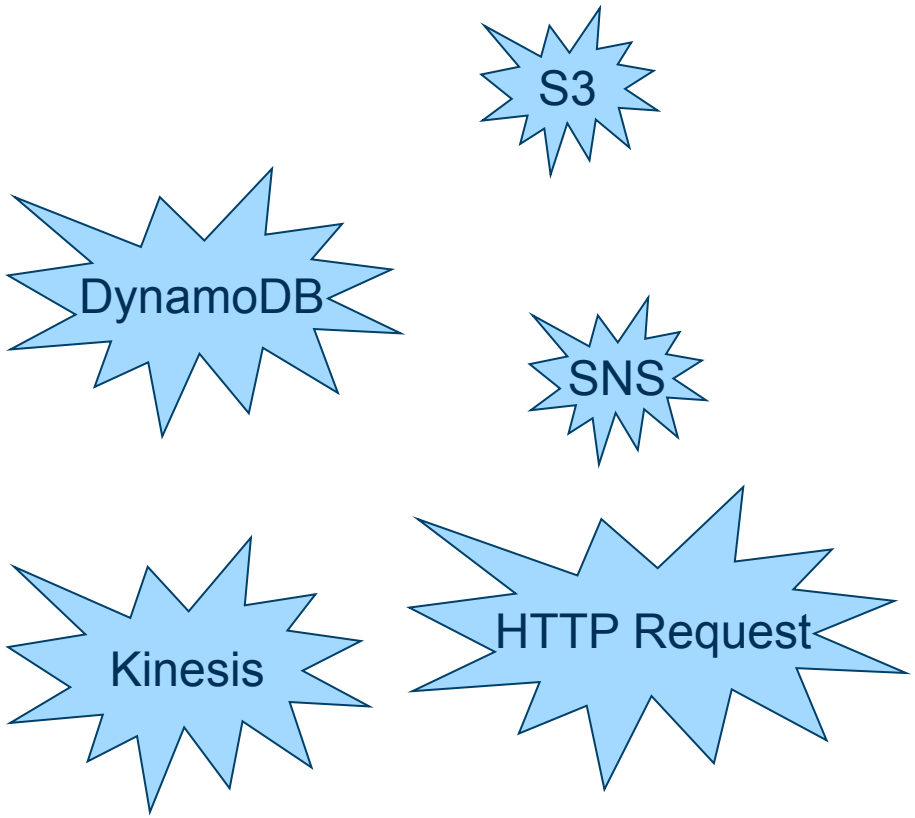
What is Amazon Lambda?

FAAS

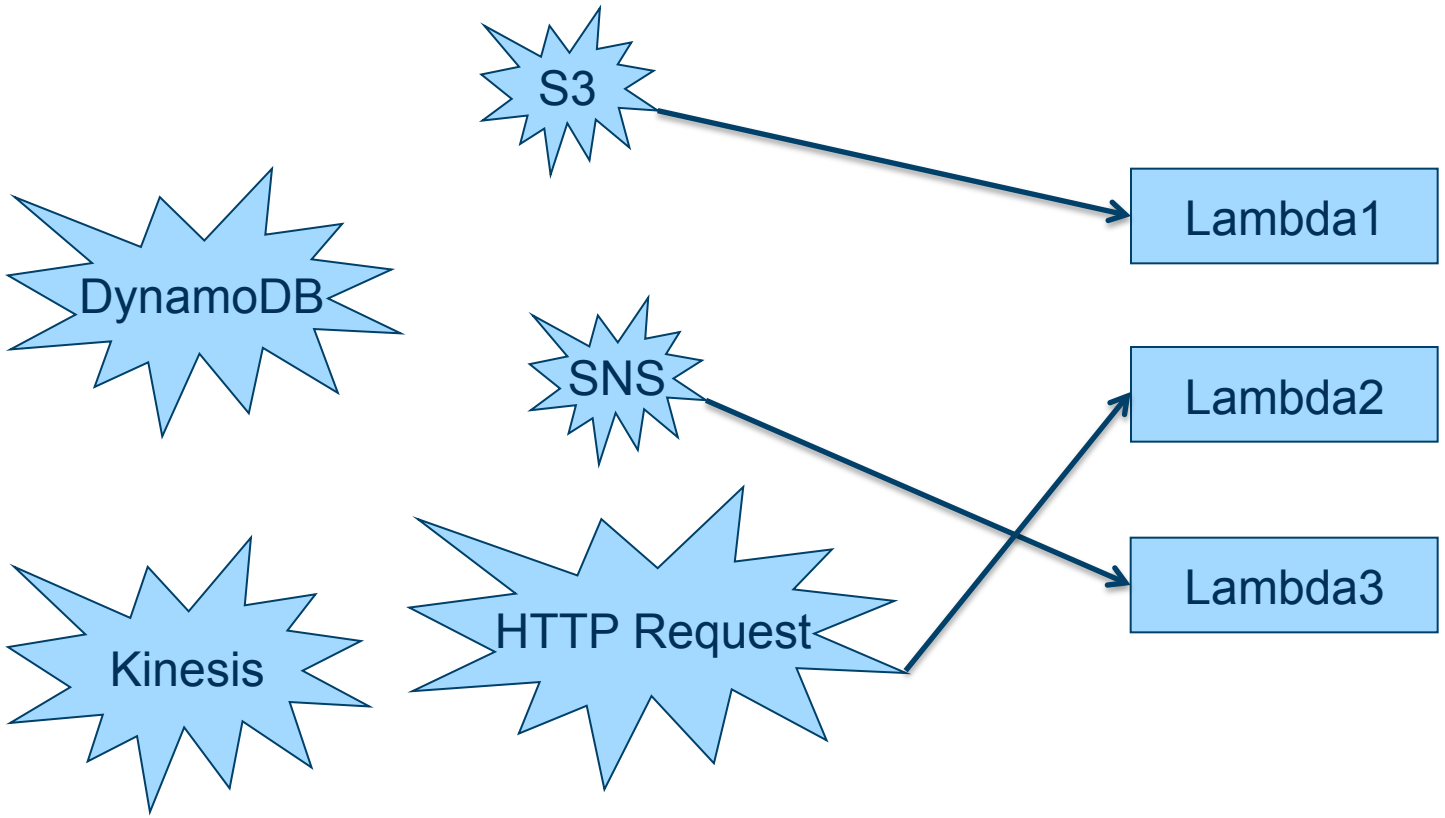
What is Amazon Lambda?

Function As A Service

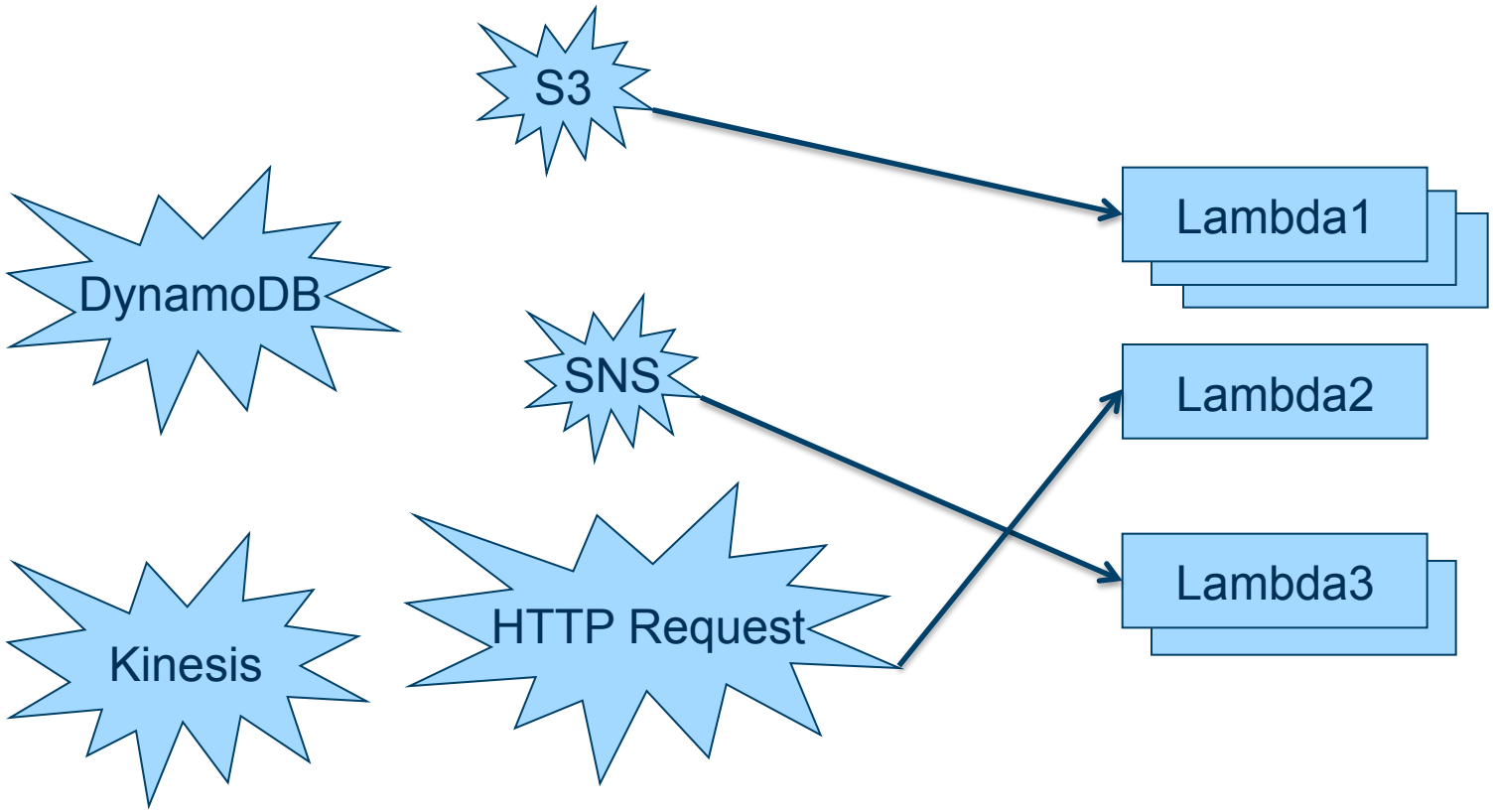
What is Amazon Lambda?



What is Amazon Lambda?

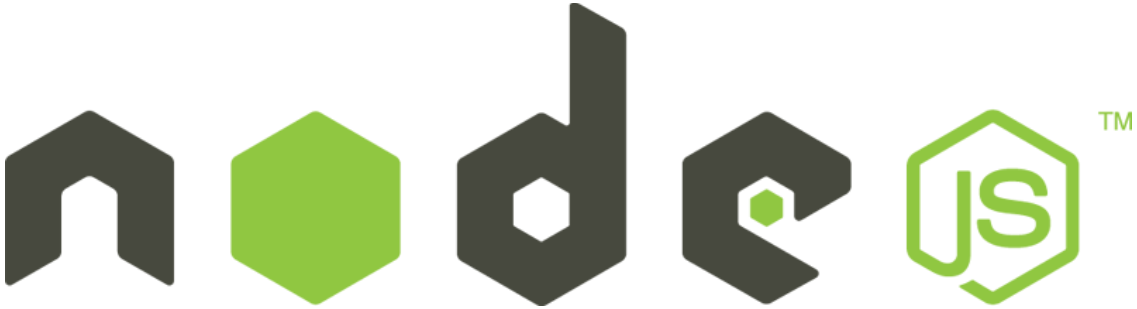


What is Amazon Lambda?



Using Amazon Lambda

Using Amazon Lambda



William HILL

Using Amazon Lambda

```
export.handler = function (event, context) {
  console.log("Processing event: " + JSON.stringify(event));
  context.success(event);
}
```

Using Amazon Lambda

```
export.handler = function (event, context) {
  console.log("Processing event: " + JSON.stringify(event));
  context.success(event);
}
```

Lack of static typing
Side-effects

Using Amazon Lambda

```
export.handler = function (event, context) {
  console.log("Processing event: " + JSON.stringify(event));
  context.success(event);
}
```

Lack of static typing
Side-effects

William HILL

Purescript

Introduction to Purescript

```
two :: Int
two = 2
```

Introduction to Purescript

```
pythagoras :: Number -> Number -> Number
pythagoras a b = sqrt (a*a + b*b)
```

Introduction to Purescript

```
pythagoras :: Number -> Number -> Number
pythagoras a b = let sqr n = n*n
                 in sqrt (sqr n + sqr n)
```

Introduction to Purescript

```
add :: Int -> Int -> Int
add a b = a + b
```

Introduction to Purescript

```
add :: Int -> Int -> Int
add a b = a + b

add2 :: Int -> Int
add2 = sum 2
```

Introduction to Purescript

```
gcd :: Int -> Int -> Int
gcd n 0 = n
gcd 0 m = m
gcd n m = if n > m then gcd (n - m) m else gcd n (m - n)
```

Introduction to Purescript

```
gcd :: Int -> Int -> Int
gcd n 0 = n
gcd 0 m = m
gcd n m | n > m = gcd (n - m) m
gcd n m = gcd n (m - n)
```

Introduction to Purescript

```
type User = { name :: String, age :: Int }
```

Introduction to Purescript

```
type User = { name :: String, age :: Int }

double :: User -> User
double u = u { age : u.age * 2 }
```

Introduction to Purescript

```
type User = { name :: String, age :: Int }

double :: User -> User
double u = u { age : u.age * 2 }

double' :: { age :: Int | r } -> { age :: Int | r }
double' u = u { age : u.age * 2 }
```

Introduction to Purescript

```
type User = { name :: String, age :: Int }

double :: User -> User
double u = u { age : u.age * 2 }

modify :: forall r. (Int -> Int) -> { age :: Int | r } -> { age :: Int | r }
modify f u = u { age : f u.age }
```

Introduction to Purescript

```
type User = { name :: String, age :: Int }

double :: User -> User
double u = u { age : u.age * 2 }

modify :: forall r. (Int -> Int) -> { age :: Int | r } -> { age :: Int | r }
modify f u = u { age : f u.age }

double' :: { age :: Int | r } -> { age :: Int | r }
double' = modify (2*)
```

Interacting with an impure world

https://github.com/lvicentesanchez/lambdaworld2015

Interacting with an impure world

```
{
  "name" : "Luis Vicente",
  "age" : 36
}
```

Interacting with an impure world

Interacting with an impure world

```
{
  "name" : "Luis Vicente",
  "age" : 72
}
```

Interacting with an impure world

```
-- export.handler = function(event, context) { ... }
handler :: ??? -> ??? -> ???
```

Interacting with an impure world

```
-- export.handler = function(event, context) { ... }
handler :: User -> ??? -> ???
```

Interacting with an impure world

```
-- export.handler = function(event, context) { ... }
handler :: User -> ??? -> ???
```

Interacting with an impure world

```
-- export.handler = function(event, context) { ... }
handler :: Foreign -> ??? -> ???
```

Interacting with an impure world

```
foreign import data Foreign :: *
```

Interacting with an impure world

```
foreign import data Foreign :: *

class IsForeign a where
  read :: Foreign -> F a
```

Interacting with an impure world

```
newtype User = User { name :: String, age :: Int}
```

Interacting with an impure world

```
newtype User = User { name :: String, age :: Int}

instance userIsForeign :: IsForeign User where
  read :: Foreign -> F User
  read value = do n <- readProp "name" value
                  a <- readProp "age" value
                  return $ User { name : n, age : a}
```

Interacting with an impure world

```
newtype User = User { name :: String, age :: Int}

instance userIsForeign :: IsForeign User where
  read :: Foreign -> F User
  read value = do n <- readProp "name" value
                  a <- readProp "age" value
                  return $ User { name : n, age : a}

type F = Either ForeignError
```

Interacting with an impure world

```
-- export.handler = function(event, context) { ... }
handler :: Foreign -> ??? -> ???
```

Interacting with an impure world

```
-- export.handler = function(event, context) { ... }
handler :: Foreign -> Context -> ???
```

Interacting with an impure world

```
foreign import data Context :: *
```

Interacting with an impure world

```
foreign import data Context :: *

foreign import failure :: ??? -> ??? -> ???

foreign import success :: ??? -> ??? -> ???
```

Interacting with an impure world

```
foreign import data Context :: *

foreign import failure :: Context -> ??? -> ???

foreign import success :: Context -> ??? -> ???
```

Interacting with an impure world

```
foreign import data Context :: *

foreign import failure :: Context -> Error -> ???

foreign import success :: Context -> Foreign -> ???
```

Interacting with an impure world

```
exports.failure = function (context) {
    return function (error) {
        return function () {
            context.done(error, null);
            return {};
        };
    };
};

exports.success = function (context) {
    return function (object) {
        return function () {
            context.done(null, object);
            return {};
        };
    };
};
```

William HILL

Interacting with an impure world

```
-- | The `Eff` type constructor is used to represent _native_ effects.
-- |
-- | The first type parameter is a row of effects which represents the
-- | contexts in which a computation can be run, and the second type
-- | parameter is the return type.
foreign import data Eff :: # ! -> * -> *
```

Interacting with an impure world

```
foreign import data Context :: *

foreign import failure :: Context -> Error -> ???

foreign import success :: Context -> Foreign -> ???
```

Interacting with an impure world

```
foreign import data Context :: *

foreign import data LAMBDA :: !

foreign import failure :: Context -> Error -> ???

foreign import success :: Context -> Foreign -> ???
```

Interacting with an impure world

```
foreign import data Context :: *

foreign import data LAMBDA :: !

foreign import failure :: forall r. Context -> Error -> Eff (lambda :: LAMBDA | r) Unit

foreign import success :: forall r. Context -> Foreign ->  Eff (lambda :: LAMBDA | r) Unit
```

Interacting with an impure world

```
-- export.handler = function(event, context) { ... }
handler :: Foreign -> Context -> ???
```

Interacting with an impure world

```
-- export.handler = function(event, context) { ... }
handler :: forall r. Foreign -> Context -> Eff (lambda :: LAMBDA | r) Unit
```

Interacting with an impure world

```
lambda :: Foreign -> Context -> Eff (console :: CONSOLE, lambda :: LAMBDA) Unit
lambda f c = let fail = failure c
                 succ = success c
                 user = modify f
             in do report user
                   either fail succ user

modify :: Foreign -> Either E.Error User
modify f = let ageX2 = modifyAge (2*)
               error = E.error <<< show
           in bimap error ageX2 $ read f

report :: forall r. Either E.Error User -> Eff (console :: CONSOLE | r) Unit
report (Left  e) = log $ "Error processing input: " ++ show e
report (Right u) = log $ "Processed user: " ++ show u
```

Interacting with an impure world

```
module.exports.handler = function(event, context) {
  return require("Main").lambda(event)(context)();
}
```


William HILL