Going serverless with AWS

Agenda

What’s Serverless

Why Serverless, part 1

Example, tools and what to expect development-wise

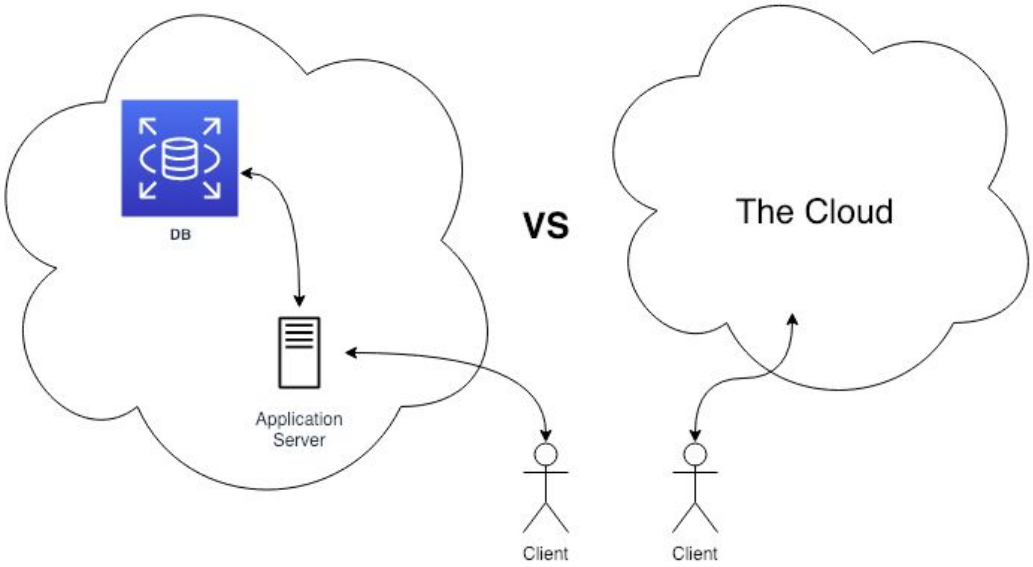
Why Serverless, part 2

Why not Serverless

Patterns for the Serverless approach

What's Serverless

My definition: lack of a persistent application server

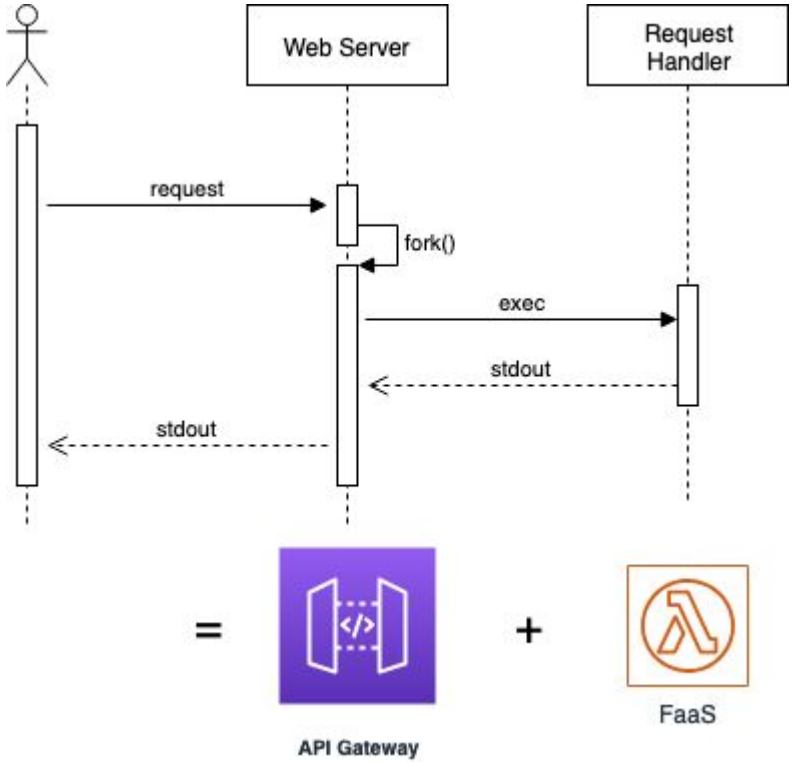


Application server MAY do:

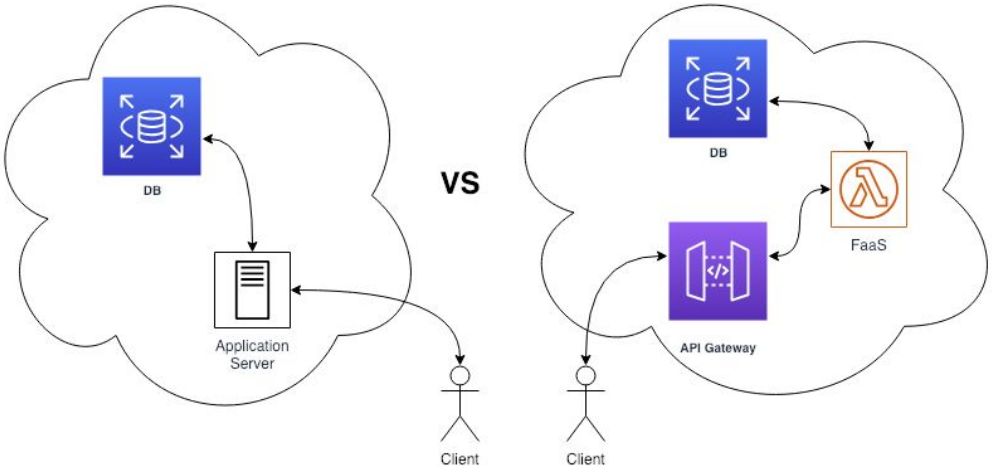
- Request handling
- State management
- Security & Auth
  -entication -orization

What's Serverless

Definitely didn't google what CGI bin is before laughing



What’s Serverless



Build an ‘application server’ from components The Cloud provides:

- Controller? - API Gateway
- Auth? - Cognito?
- State? - ElastiCache

Know ‘The Cloud’s capabilities, you’ll have to stitch them together

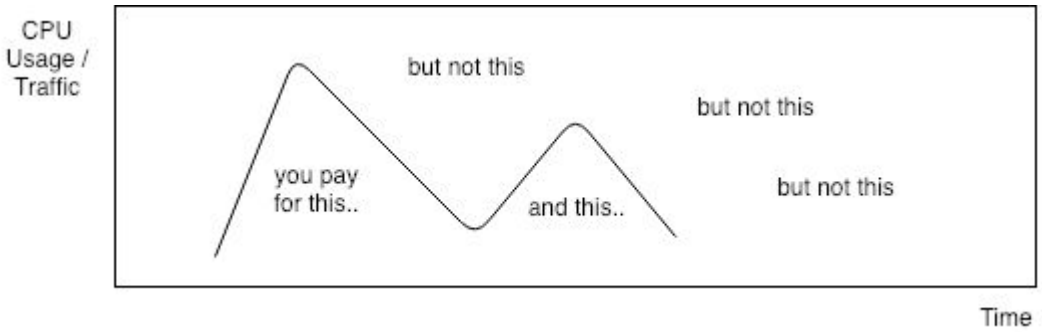
What’s Serverless

Ref: Application Lifecycle Management in a Serverless World

- Simple but usable primitives
- Scales with usage
- Never pay for idle
- Availability and fault tolerance built in

Why Serverless, part 1

For now.. it’s cheap, see * ? https://aws.amazon.com/lambda/pricing/



We’ll see more of ‘why’ later

* unless you follow some anti-patterns

Simple (and somewhat stupid) example

“Memoisation”

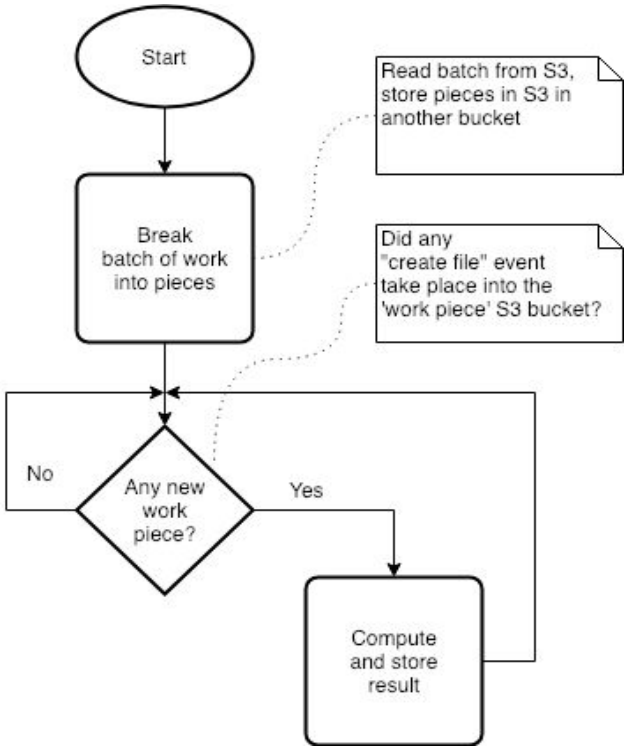
Given a batch of work to be done, compute and store the result.

```
{
   "ops": [
      {"name": "1_plus_1", "op": "1+1"},
      {"name": "2_by_2", "op": "2*2"}
   ]
}
```

->

```
1_plus_1 -> 2
2_by_2 -> 4
...
```

Simple example



Simple example - the infrastructure

With the serverless toolkit:

```
...
    compute:
        handler: serverless_hello/worker.compute
        events:
            - existingS3:
                bucket:
${self:provider.environment.PROC_BUCKET}
                events:
                    - s3:ObjectCreated:*
                rules:
                    - suffix: .work
...
```

See the full serverless.xml file

Have to declare:

- Infrastructure resources (eg. S3 buckets)
- Event handlers (HTTP, S3 .. lots and lots)

Why this makes sense: your application is not ‘just code’ !

Simple example - the code

Write handlers code:

```
...
def start_work(event, context):
    """ starts work on a series of tasks;
    each line in an input file becomes a task,
    a file of its own in a separate bucket
    """

    in_work_s3_object = get_object_from_s3(WORK_BUCKET, 'work.json')
    ops_bulk = in_work_s3_object['Body'].read().decode('utf-8')
…
```

See the full source

All lambda functions take a context and event (a dict) parameters

Simple example - the testing

Unit tests - the usual

Integration tests:

- start a local environment (that emulates AWS services) or,
- deploy to AWS (preferred)

Note how you can deploy to any number of environments - every dev may get their own!

SDLC highlights

- Have to understand ‘The Cloud’ and its services
- Have to think your application’s architecture to be ‘event-driven’
- Testing - just feels ‘different’ with limited capabilities to run locally

Why Serverless, part 2

Small operational costs
...is mostly managed by ‘The Cloud’

It’s quick for a small (web) service / app
...and you might design your big app as a collection of small services

Less to worry about infrastructure
...is mostly managed by ‘The Cloud’

Availability built-in
...is mostly managed by ‘The Cloud’

Small operational costs
...is mostly managed by ‘The Cloud’

Why not Serverless

Vendor lock-in & control
‘The Cloud’ drives how infrastructure and base services work

Testing
No very good tools for running locally - how to do stubs, tear-up, tear-down..

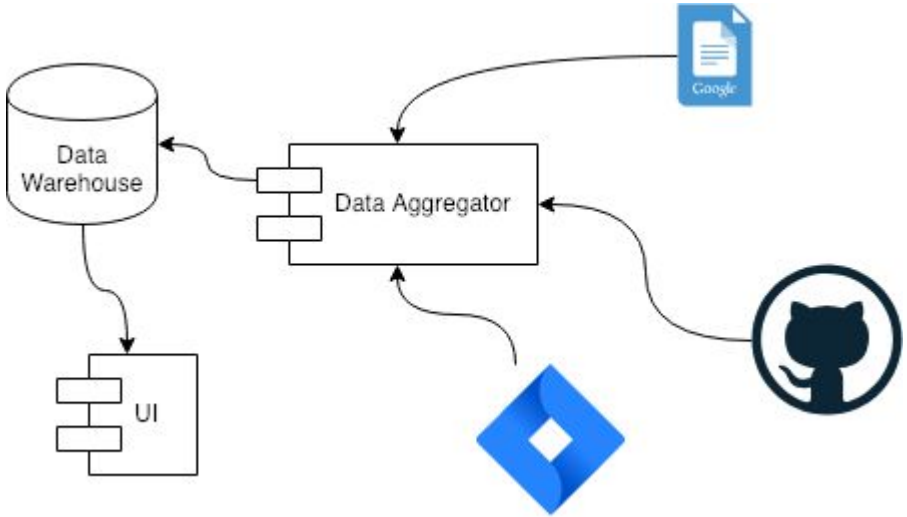
Debugging
Have to rely pretty much on what ‘The Cloud’ provides

There’s progress on improving on all of these!

Eg. https://github.com/thoeni/aws-sam-local

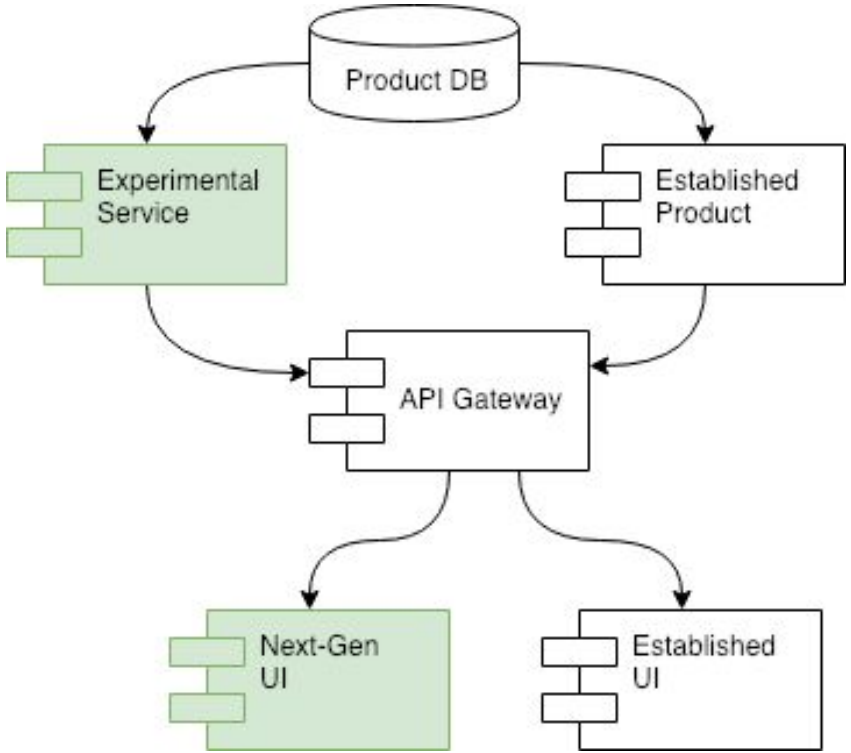
Patterns for the Serverless approach

API stitch-up job, thick UI



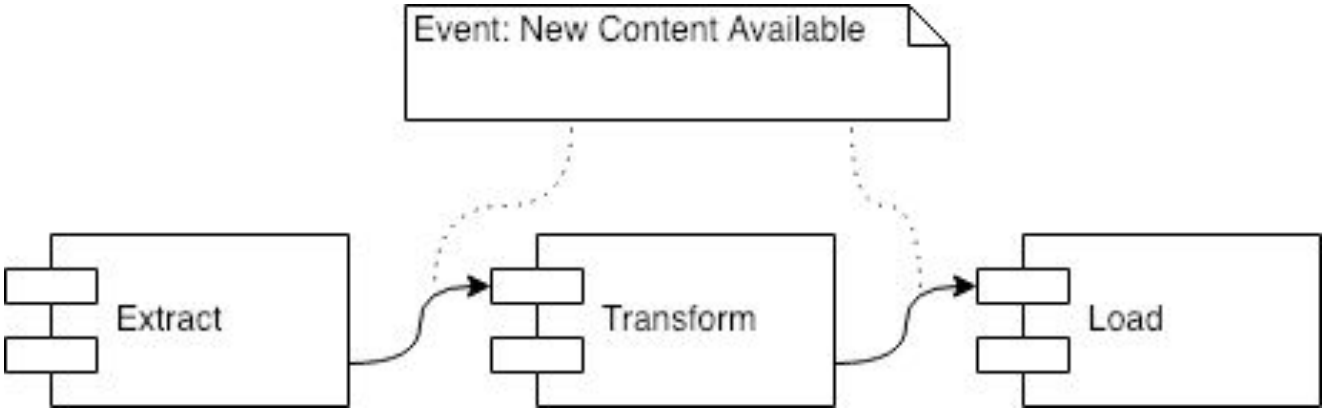
Patterns for the Serverless approach

Experiments



Patterns for the Serverless approach

Data Processing Pipes (suitable because event-driven support)



Serverless Anti-Patterns

Server-as-a-function
All your application as a single function, see zappa
- Start-up time will be big
- It's not going to be cheap anymore

Too many, too granular functions
- Hard to manage (eg. version deps) and monitor
- Excessive communication

Functions calling other functions
- Won't be cheap anymore
- You're blurring the line between many domains of concern - think about error and scaling isolation
- Functions should do one thing only!

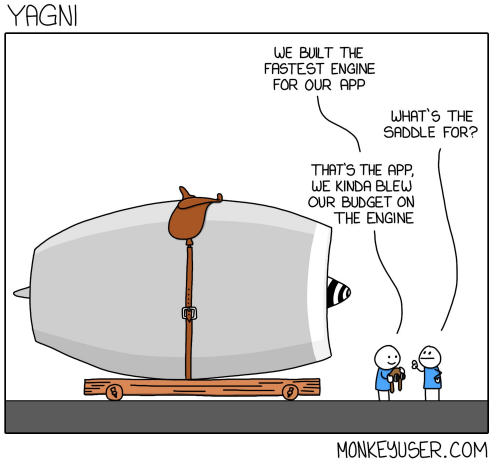
Long-running functions
- Won't be cheap anymore
- There's a 'timeout' - 5 min

..and anything that is getting close to an established limit
See https://docs.aws.amazon.com/lambda/latest/dg/limits.html

Should I learn it?

Yes, give it a try - it will only evolve in adoption, because of business value:

- Small operational costs
- Fast time-to-market (see ‘patterns’, stitch-up jobs are very common when everything has an API)



AMA

Where’s the sample code? https://github.com/QCatalyst/ro-python-serverless

How to get started?

Other mentionable tools? Eg. https://github.com/aws/chalice, https://github.com/localstack/localstack