AWS Lambda & your Majestic Monolith

AKA: Serverless for those of us with servers

Luke Closs - @lukec - CTO / Founder

ReCollect

Goal of this talk

ReCollect

Help you understand:

- What Lambda is
- Why you should care
- How it's helped us at ReCollect
- Challenges / Lessons

First - About Me

ReCollect

Software developer of 20 years

- 2G/3G systems, Enterprise Gateway Security, Enterprise Social Software, ….

Founder / CTO of ReCollect.net

- Communications platform for cities

About ReCollect's Infrastructure

ReCollect

Mix of new & old tech, evolved over ~10 years:

- EC2 Servers running REST APIs, Job Queue
- RDS / PostgreSQL database for most app data
- Majestic Monolith serves most app functions
- Handful of smaller services that support core app

So what is AWS Lambda / Serverless?

ReCollect

- It's not a big deal
- It's kinda a big deal

Serverless:

- It's a general term describing an architecture
- It's also a specific toolset
  https://www.serverless.com/

It's not a big deal - it's simple

ReCollect

Harkens back to the good ol' days of cgi-bin:

- Write some code
- Zip it up
- Upload it to "the cloud"
- Your code magically runs
- Multi-language

Simple Example: Python

ReCollect

```
def lambda_handler(event, context):
  param = event['Param1']
  return {
    "Foo": param
  }
```

- Your code gets an event
- You return a result.
  - (and/or an error!)
- Your code can be as complex as you need.

It's not a big deal - trigger it as needed

ReCollect

- When does your code run?
  - To serve HTTP requests? (API Gateway)
  - On a timer (Cloudwatch events)
  - When something happens (eg: S3 Trigger)
  - When you trigger it? (eg: API call from app)

OK, but also IT'S KINDA A BIG DEAL

ReCollect

- I can focus on my code instead of deployment details. (moving up the stack)
- I can easily deploy code in different languages.
- Useful tool in my toolbox for some problems.
- I only pay for what I use. (Tradeoffs here!)
- Scaling? Tuning? Monitoring? Logging?

It's kinda a BIG DEAL - FinOps

ReCollect

Server-world

- We're paying for EC2 servers, all our code runs there. Costs & Compute are lumped together.

Serverless-world

- Per-function billing
- Revenue based development!!!
- Is it cost effective to refactor this?!?

ReCollect

Lets talk specifics.

4 ways Lambda was helpful.

How Lambda has helped ReCollect

ReCollect

1. Offloading work from core app
2. Edge CDN routing
3. Event-driven triggers
4. Architecture Extension / Integration points

ReCollect: Offloading work from App

Context: As our app scaled, some parts became bottlenecks. One category was slow API endpoints.

Usually you look to optimize these APIs using various techniques like caching, indexing, …

But two were challenging: PDFs & Image resize

ReCollect: Offloading PDFs from App

Old Style

- Handled like any API request to our backend.
- Need to scale up API servers to add capacity.
- Calls out to `wkhtmltopdf`

New Style

- AWS Lambda, Golang
- Separate small codebase
- Auto-scales to ~infinity
- ~similar performance
- Still using `wkhtmltopdf`

Lambda PDF Benefits

ReCollect

We instantly stop worrying about scaling &concurrency.

- Costs are directly quantifyable (and marginal!)
  166k executions * 4000 ms average @3G memory
  =a whopping $25/month (+~$20/mo for API Gateway)
- Offloads work from main API, so more capacity there!
- Small &easy to build &test. “Set it and forget it”

Contrast this against effort spent trying to scale up the main app for this use-case.

ReCollect: Edge CDN Routing

ReCollect

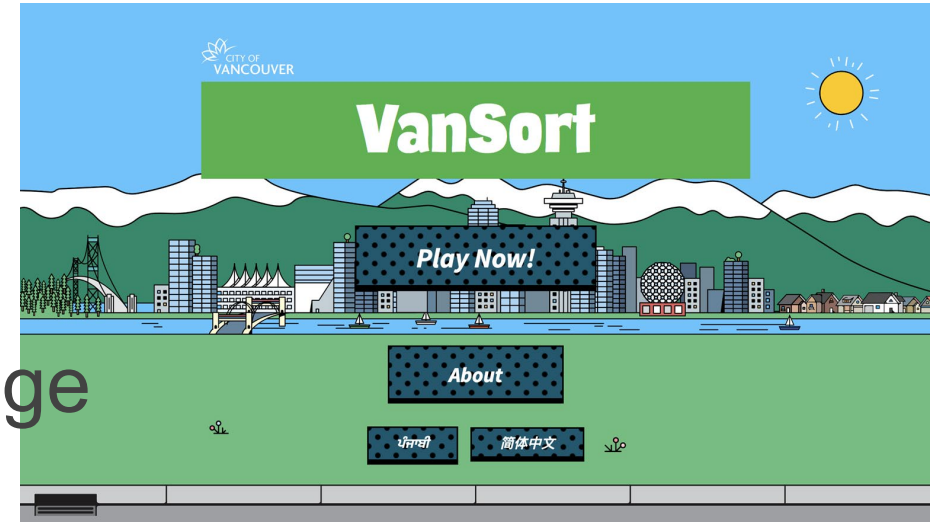
Context: ReCollect makes an educational sorting gam

It's served as static assets from a S3 bucket.

vancouver.recycle.game

We wanted clean URLs to map into our S3 bucket.

Cloudfront + Lambda @ Edge



ReCollect: Edge CDN Routing

ReCollect

```
exports.handler = (event, context, callback) => {
    var request = event.Records[0].cf.request;
    var domain_rx = /^(.+?)\.(?:staging\.)?recycle\.game$/;
    var host = request.headers['host'][0].value || '';
    var match = domain_rx.exec(host);
    var area = match[1];

    var orig_uri = request.uri
    var uri  = orig_uri.replace(/^\/index\.html$/, '/');

    if (uri.match(/^\/(?:[a-z]{2}(?:-[a-z0-9]{2,4})?\/?)?$/i)) {
        var new_uri = "/dist/cities/" + area + uri + 'index.html';
        request.uri = new_uri;
        console.log(`Rewrote URL ${orig_uri$} to ${new_uri}`);
    }
    callback(null, request);
};
```

Edge CDN Routing Benefits

ReCollect

Less infrastructure to worry about.

Super easy to build automated tests to verify correctness.

Because our team was familiar with AWS Lambda, we could quickly build a low-maintenance solution.

Cost is negligible. (Low memory + fast execution time)

ReCollect: Event-driven triggers

Context: Our system processes updates from files in a S3 bucket. Previously we polled for updates.

We wanted to make this more real-time.

AWS Lambdas can be triggered based on S3 events.

ReCollect: Event-driven triggers

ReCollect

Old Style

- Core app has code + cron
- Checks files in S3
- Must maintain state about the file to detect changes
  ~OR~
  Blindly runs each time.

New Style

- s3:ObjectCreated* trigger
- Lambda, NodeJS 10
- Serverless framework
- Triggers update via our API when files change.

Event Driven Trigger - serverless.yaml

ReCollect

```
service: s3-connector-hook
provider:
  name: aws
  runtime: nodejs10.x
  region: us-east-1
  timeout: 3

functions:
  conn-hook:
    handler: conn-hook.run
    events:
      - s3:
          bucket: recollect-data-bucket
          event: s3:ObjectCreated:*
          existing: true
          rules:
            - prefix: prod/
            - suffix: .json.gz
```

Our team has really enjoyed using the serverless framework for developing, debugging, deploying and managing our lambdas.

The serverless.yaml describes our functions and how they are triggered. In this case, we want to run this function when certain files are uploaded.

(Note: I've omitted some exciting IAM permissions for brevity.)

Event Driven Trigger - NodeJS

ReCollect

- Receives an event describing the file that changed
- Lambda calls into our main app via REST API
  - Triggers the data processing
  - This API already existed, so no additional work!
- Small codebase, easy to test & debug.

ReCollect: Extensible Integrations

ReCollect

Context: Our system can forward requests into lots of other systems, typically via REST API.

But every system is different, require custom code.

Every new integration, we were adding more code to our backend APIs. How can we lower the cost to build new integrations & also reduce code bloat?

ReCollect: Extensible Integrations

ReCollect

Thinking about the problem:

- Some integrations would be much easier in other languages. Eg Freshdesk has a nodejs library we could use, but our backend is not nodejs…
- Integrations differ in 2 ways:
  - How to map our data to the other system's
  - How to deliver that data into their system (may fail)
- Minor changes to integrations require deploy of full system…
- We're planning to add more integrations, where are we headed?

ReCollect: Extensible Integrations

ReCollect

So we defined a state machine using AWS Step Functions to define how integrations work.

Each integrations needs to define 2 Lambdas:

- PrepareRequest
- SendRequest

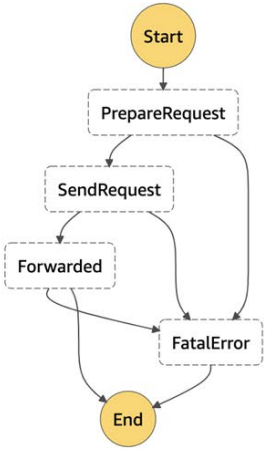
The other 2 steps are handled by Lambdas shared between all integrations.

Each Lambda can be implemented in the most convenient language.

Definition

Export ▼ Layout ▼

```
{
  "Comment": "ReCollect forwarding integration for ForwardSupportRequestToFreshDesk",
  "StartAt": "PrepareRequest",
  "States": {
    "PrepareRequest": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:103772702347:function:ForwardSupportRequestToFreshDesk_PrepareRequest",
      "Next": "SendRequest",
      "ResultPath": "$.prepared",
      "Retry": [
        {
            "ErrorEquals": ["States.TaskFailed"],
            "IntervalSeconds": 60,
            "MaxAttempts": 5
        }
      ],
      "Catch": [
        {
            "ErrorEquals": [ "States.ALL" ],
            "Next": "FatalError"
        }
      ]
    },
    "SendRequest": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:103772702347:function:ForwardSupportRequestToFreshDesk_SendRequest",
```



ReCollect: Extensible Integrations

ReCollect

Old Style

- All code lived in core app, was growing &bloating.
- Integrations must be implemented in core language
- Required knowledge of existing core system.

New Style

- Each integration has simple, separate implementation.
- PrepareRequest is dead simple, clear &easy to test.
- Very clear requirements, could even outsource dev.
  (Get a hash with data, send it.)
- StepFunctions handles retries

Lets ReCap

To ReCap: Lambda / Serverless

ReCollect

It's not a Big Deal

- Pretty easy to use, doesn't require huge investment.
- Simple things are simple.
- Easy to get started - get started on something non-mission critical.

It's a Big Deal

- Where the future is headed
- Introduces Finance to Dev
- A powerful tool in an Architect / CTO's toolbox.
- A new set of emergent practices are forming to eventually replace DevOps.

What you should be thinking about

ReCollect

Experimentation:

- Find small projects your team can learn these skills on.
- Learn what parts of your systems would and wouldn’t be appropriate for serverless today.
- You will need to learn new (emergent) practices about topics you already deal with: Debugging, Logging, Monitoring, …

What you should be thinking about

ReCollect

Skill Development:

- Debugging- how is debugging different in this world?
- Logging- how is logging different from what you're used to?
- Monitoring - how do you know it's still working?

These are (hopefully) all questions you know how to answer today with your (monolith?) app.

Learning new things can feel clumsy at first. This is the same!

Remember: Best Practices are still emerging

What you should be thinking about

RecCollect

Maintenance:

- It’s not 0! But close. So close you’ll forget how it works!
  - Eg: nodejs 8.x deprecation-> nodejs 10.x

- Solution: Document your (micro)systems!
  - Every system we build has a wiki page that describes the architecture, how to change, test, debug, …

What you should be thinking about

ReCollect

Finance implications:

- Should I refactor this code if it's only going to save us $8.42 per month but takes me 5 hours? (probably not.)

- This lambda is now costing me $X00 per month - lets allocate Y developer hours to optimize that.

BONUS: MAPS!

ReCollect

Maybe you don't think serverless is the future yet.

Let's look at some maps.

Bonus Section - Wardley Maps

ReCollect

As a super brief overview of Wardley Maps, - this is a map of a Tea Shop that uses custom built kettles.

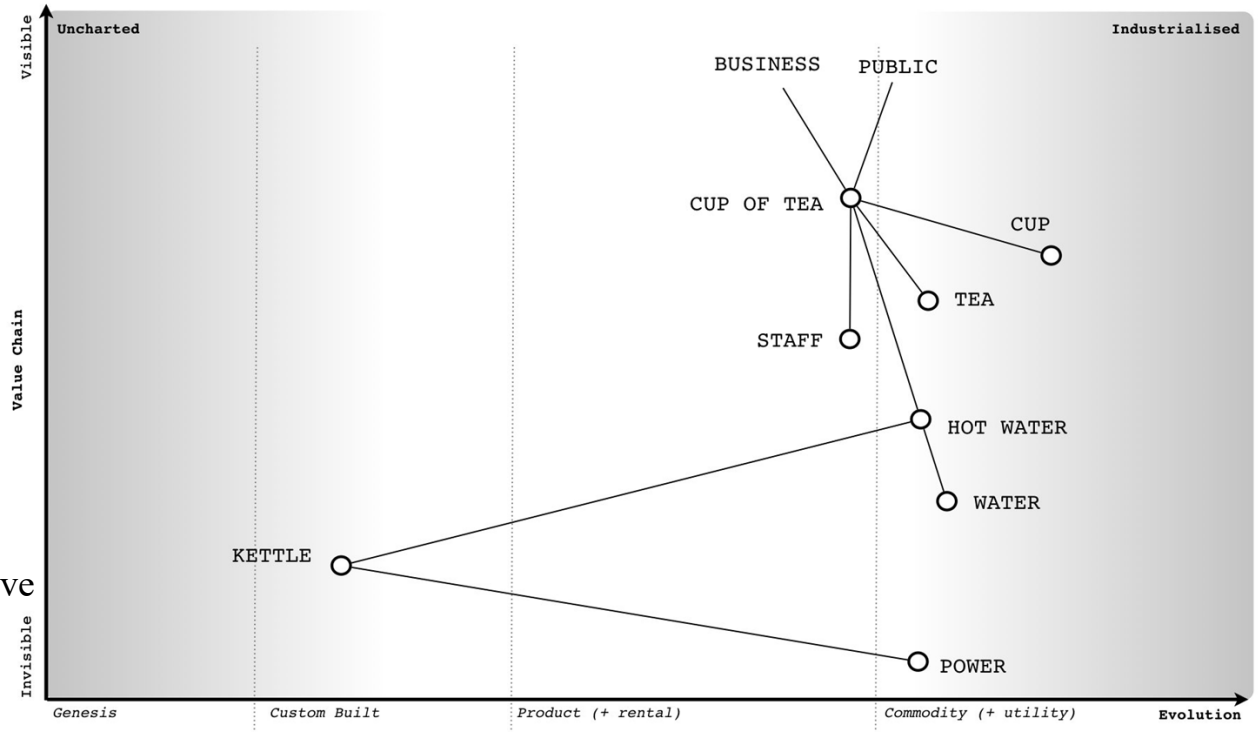
Users at the top.
Show components / dependencies
Least visible to user is at bottom.

Left to Right indicates Evolution
- Left side is invention,
- Then custom built,
- Then Products
- Then Commodity / Utility

Maps are communication devices!

Probably doesn't make sense to have Custom kettles, right?

Map credit @swardley.



Bonus Section- Pre-cloud Map

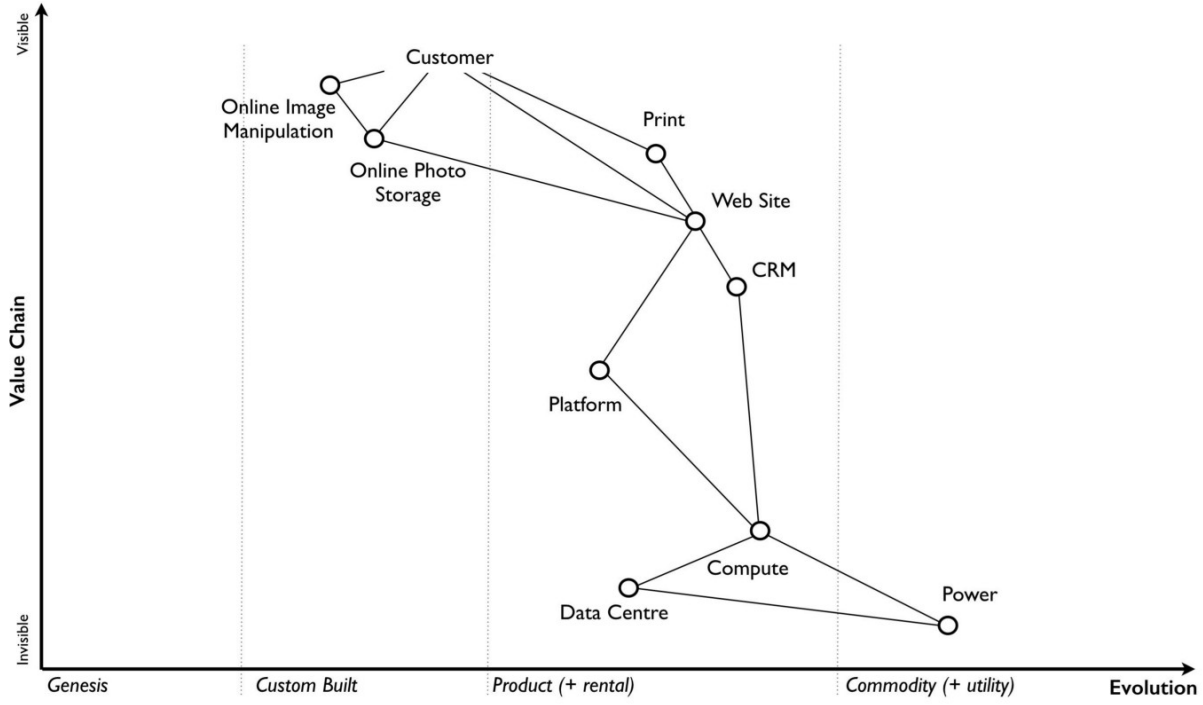
ReCollect

This is a map of a image web app, back in the day, pre-cloud.

Remember back when we ran our own Servers in our own data centres?

Some readers' careers never included hosting their own data centres. Other readers remember racking their own servers. And a few readers still have their own data centers still.

It was a pain to have lots of servers, so we tended to Scale UP by buying ever more beefy servers.



Map credit @swardley.

Bonus Section- Dev-Ops is born

ReCollect

But as some teams began to Scale OUT, they started running hundreds or thousands,... of machines.

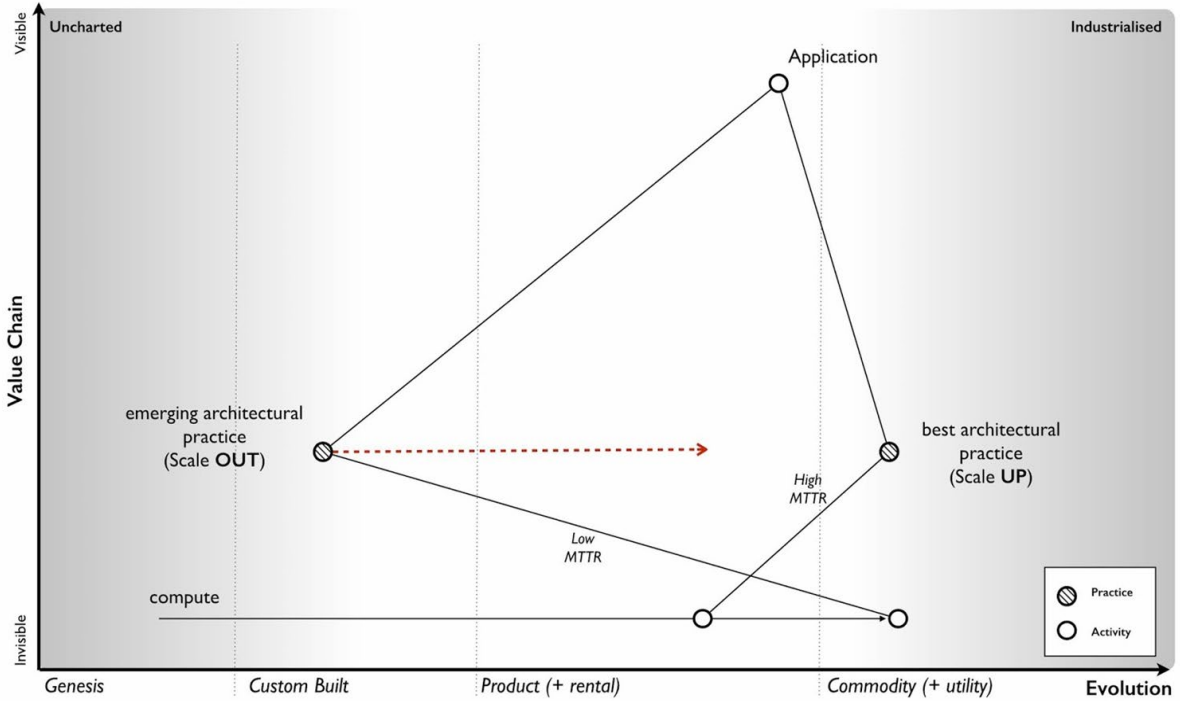
And then this wild crew called "DevOps" Started advocating we treat our lovely Servers as "Cattle, not Pets".

They said developers should manage infrastructure instead of the Ops Team.

They built new practices such as Infrastructure as Code.

Today DevOps doesn't seem so crazy.

Map credit @swardley.



Bonus: A new practice is coming…

ReCollect

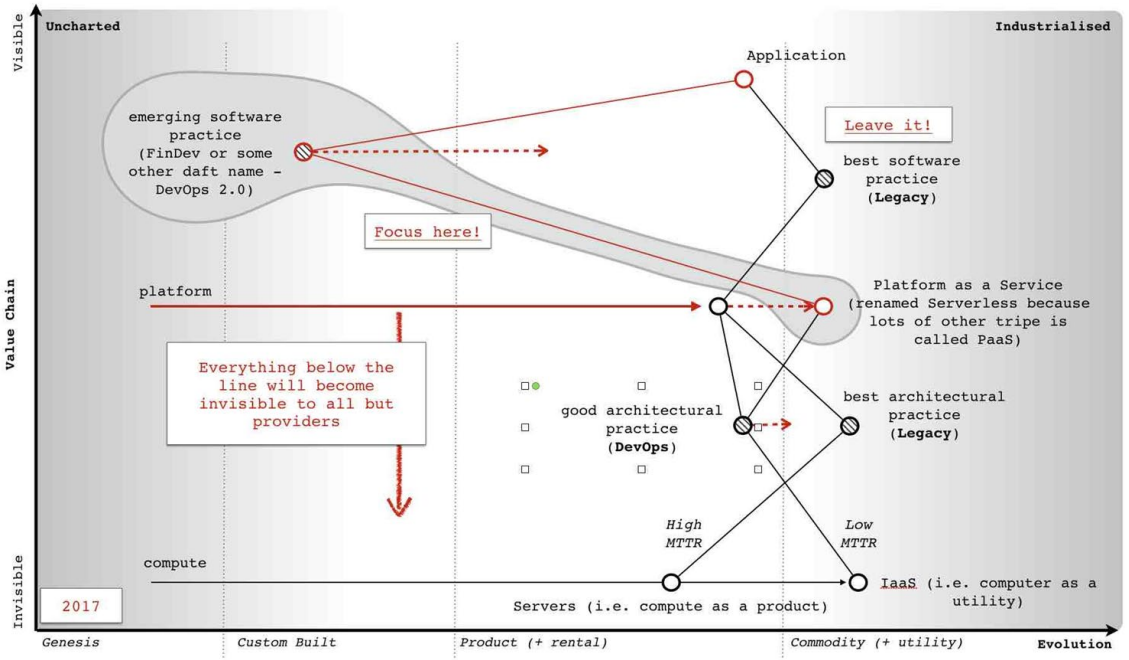
But what's better than managing lots of servers? Not managing servers.

Serverless / Lambda tech hopes to move us another step up the stack, so we can focus on delivering more value.

So new architectural practices are emerging.

Today some of this seems crazy and hard to imagine, because of our current Perspective.

I think it's worthwhile to start building capacity in your teams around these emerging practices.



Map credit @swardley.

Bonus: A slide from 2030

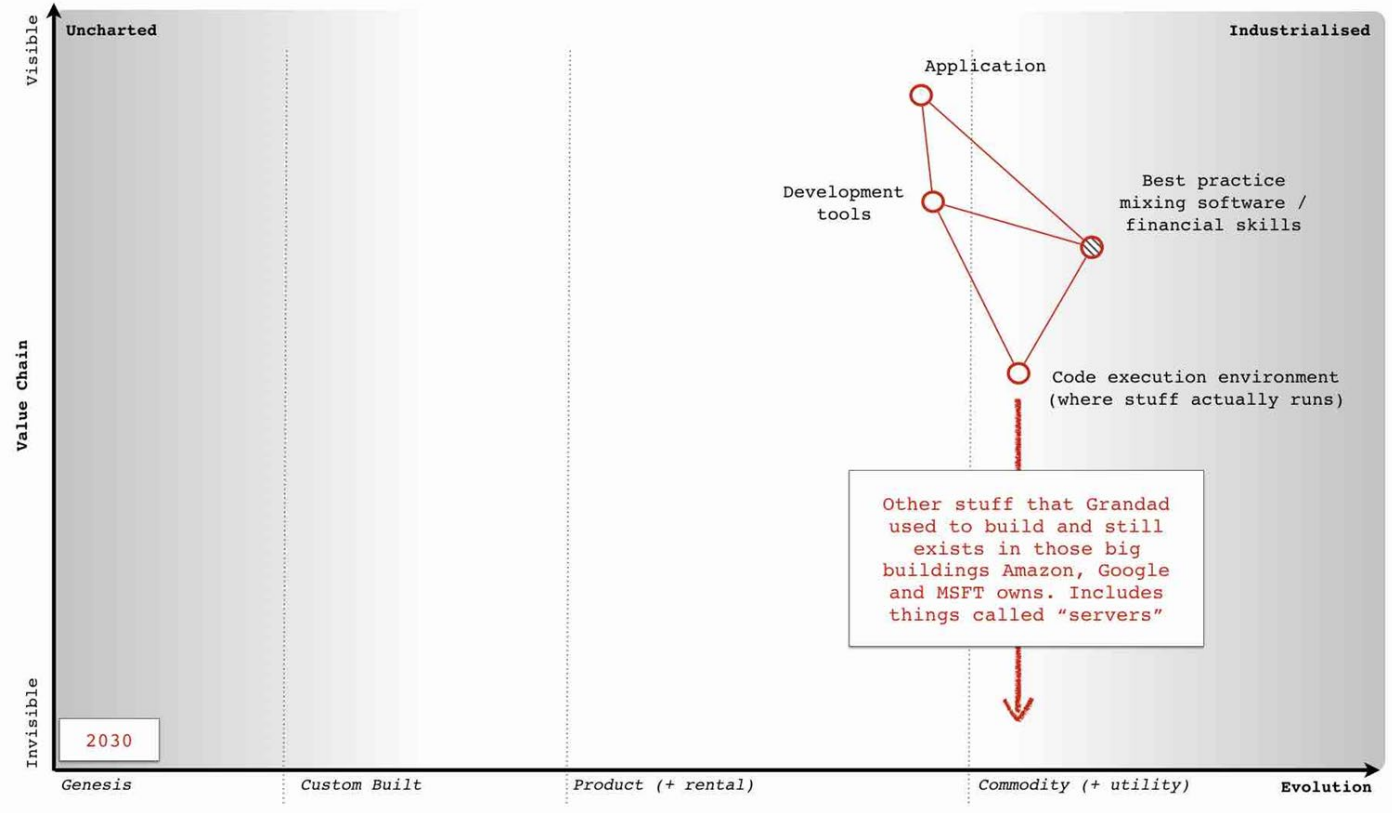
ReCollect

"Remember having to run your own servers?!"

What tools are we going to use in 10 years?

What skills should our teams start developing?

What architectural choices are you making now that have lasting implications?



Map credit @swardley.

Luke Closs @lukec - luke@recollect.net

Thank you! Happy to take questions!

Wardley Maps

All images credit Simon Wardley - @swardley

https://en.wikipedia.org/wiki/Wardley_map

https://medium.com/wardleymaps