Chris Priest

@cjrpriest

ALTERNATIVE WAYS TO THINK ABOUT SERVERLESS

Hi,

Welcome to my talk on Alternative Ways to Think About Serverless

My name is Chris Priest & I work for the cloud-first consultancy Amido

We’re hiring all sorts of people who want to work on cloud-based software, so come and talk to us if you are interested!

A Story
About Bob

https://flic.kr/p/c8KfNj

Going Fieldless

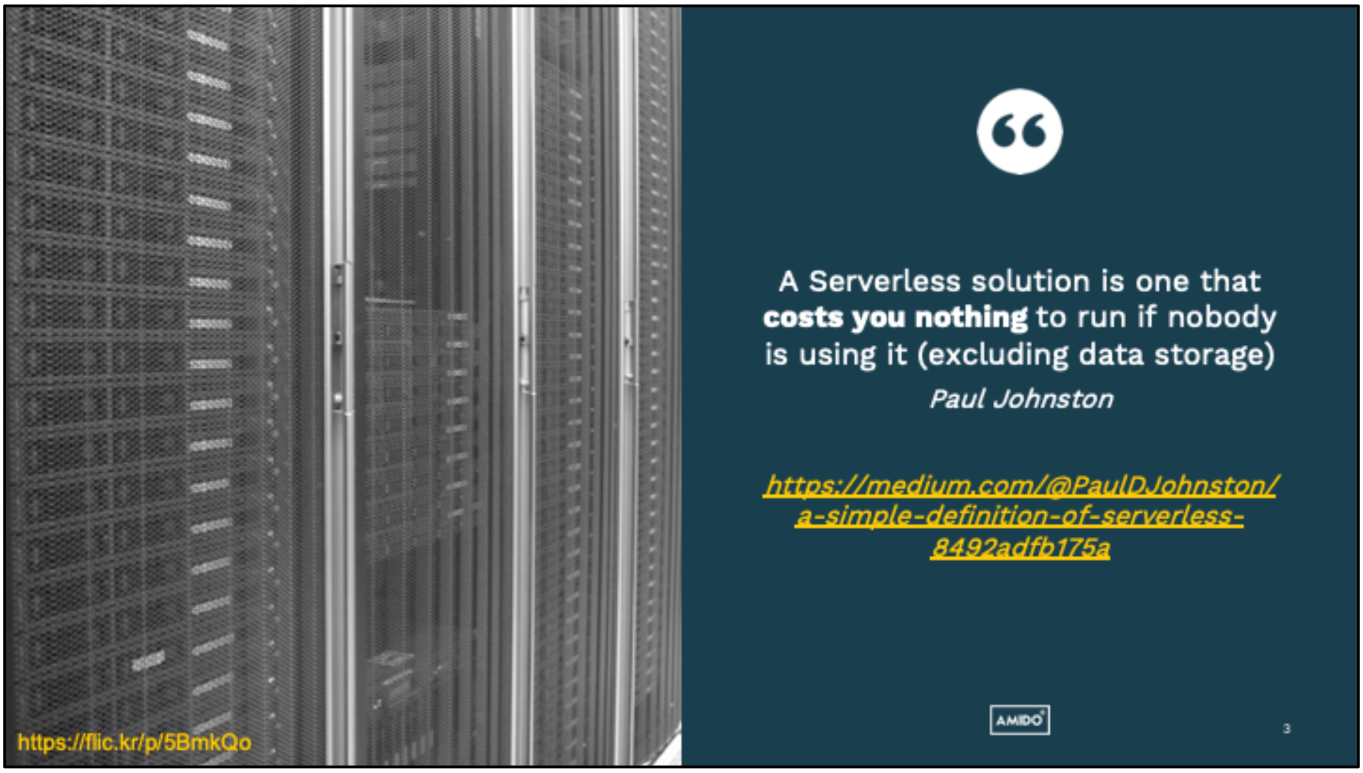
- Meet bob, it’s some point in the distant past. Bob, like a lot of his friends, has his own field. He ploughs it, plants vegetable seeds, grows his vegetables, harvests them, and provides for his family.
- He has been down the local drinking establishment and he’s heard of this new way to get vegetables to feed his family. And it’s been termed “Fieldless”.

- He doesn’t think it’ll catch on — I mean, can he really trust someone else to grow his food for him? Can he really guarantee the security of his food?

- Nonetheless, bob is going to give it a try. He goes to his local vegetable trader, pays just for what he needs and takes it home with him. Now he doesn’t need to worry about his field at all. The time he did spend tending to the field, he can now spend doing other higher value things, such as spending time with his family, or learning specialist skills that earn him more money.
- He’s delegated all of the concerns of growing food (ploughing, seeding, harvesting the field) to a third party who frankly is much better at doing it. They have the experience and the scale to do it better, and cheaper.

Segway:

- It seems clear to me that over time, in history, we have sought to delegate un-differentiating heavy lifting to third parties who are more capable / willing to do that. Whether it be the basics like provision of food & shelter, or more more advanced concepts such as building cars or providing medical help.
- Should the cloud be any different?
- For me, this move to remove un-differentiating heavy listing is inevitable — in the future programming against anything that other than a serverless model will feel as weird as programming against a mainframe does now



https://flic.kr/p/5BmkQo

> A Serverless solution is one that **costs you nothing** to run if nobody is using it (excluding data storage)
>
> *Paul Johnston*

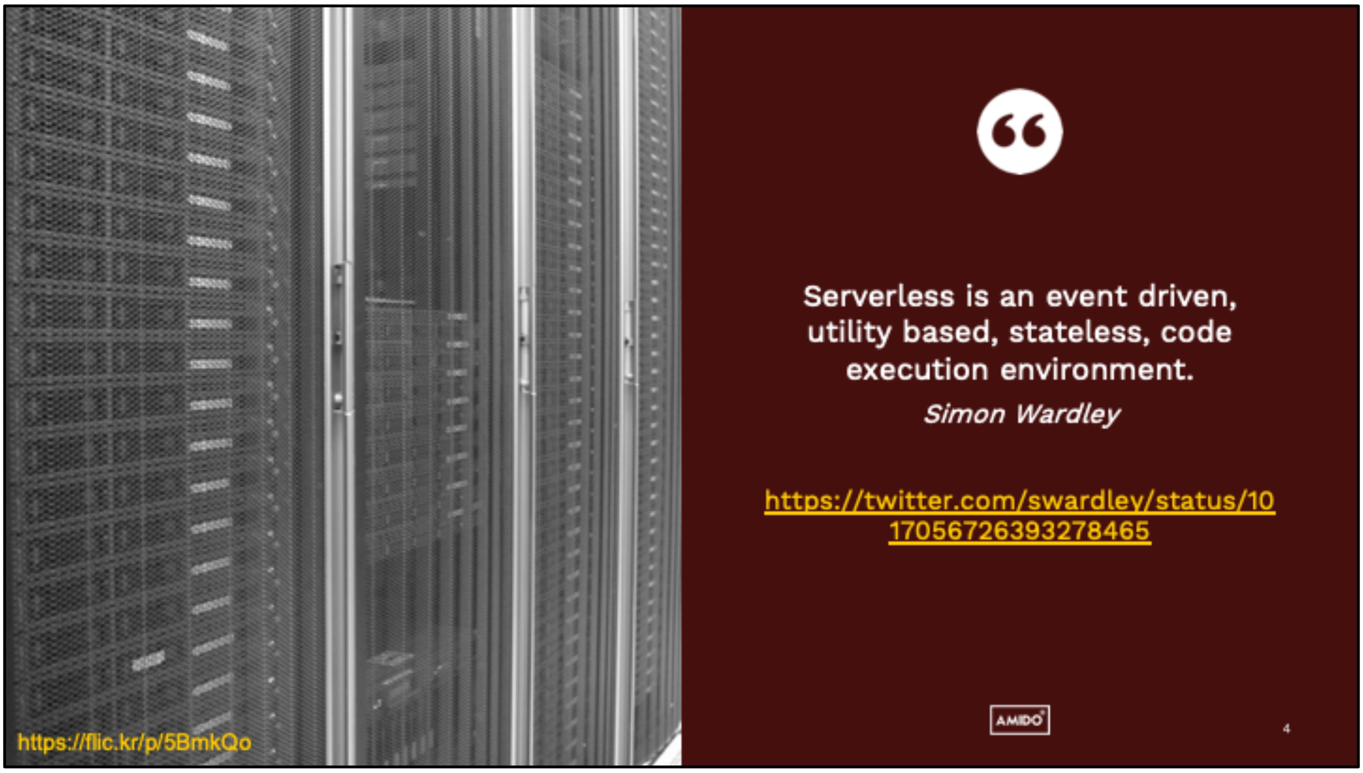
https://medium.com/@PaulDJohnston/a-simple-definition-of-serverless-8492adfb175a

AMIDO

3

Lets talk about some of the definitions of serverless that are out there.

- “A Serverless solution is one that costs you nothing to run if nobody is using it (excluding data storage)”
- I believe this is broadly correct, and it's great one-liner, place to start, but there is a huge amount of value in the operational side that is not encapsulated here.



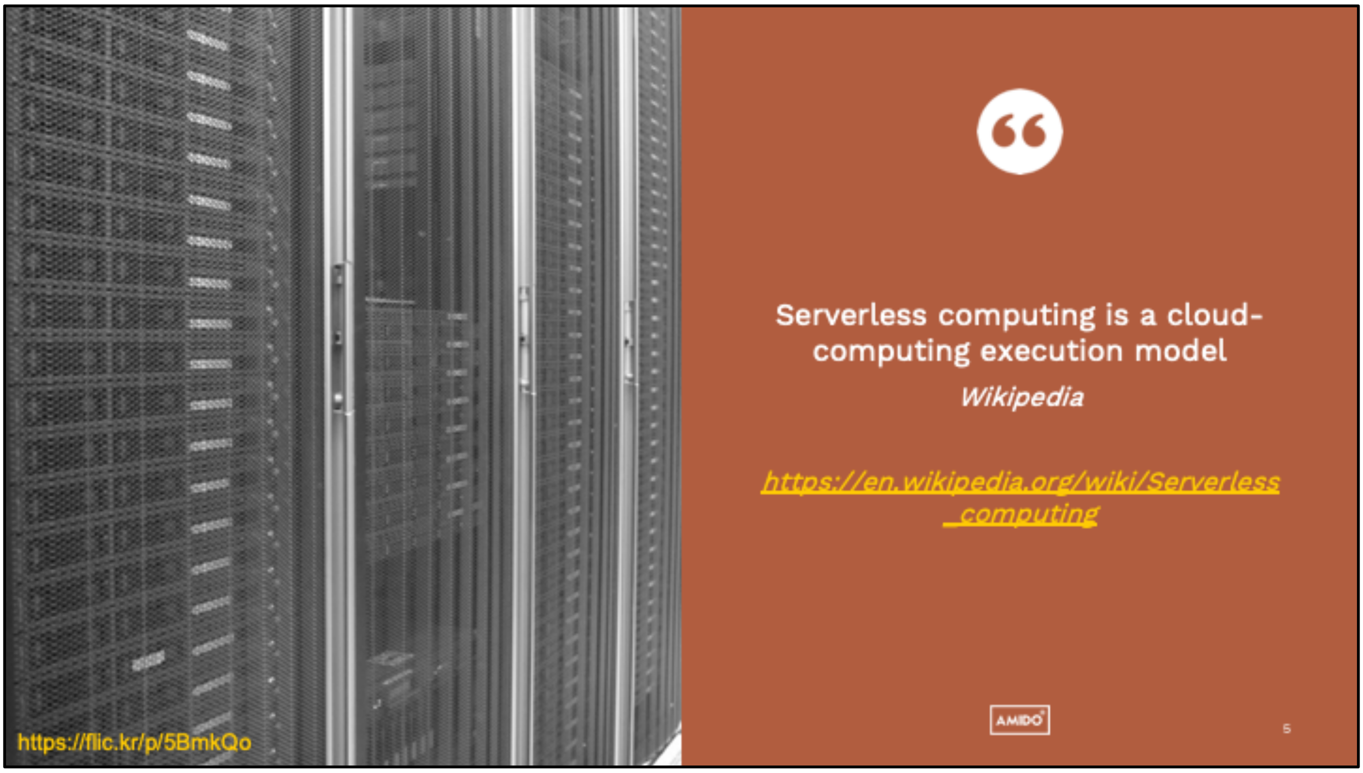
https://flic.kr/p/5BmkQo

> Serverless is an event driven, utility based, stateless, code execution environment.
>
> *Simon Wardley*

https://twitter.com/swardley/status/1017056726393278465

4

- “Serverless is an event driven, utility based, stateless, code execution environment.” (Ref https://twitter.com/swardley/status/1017056726393278465)
- He also goes on to say: if you wanted to make some toast, would you buy a toaster for $40, or spend thousand of dollars making a toaster from scratch using raw materials
- The same should apply to executing code — why would you not leverage existing platforms as your execution environment?



https://flic.kr/p/5BmkQo

> Serverless computing is a cloud-computing execution model
>
> *Wikipedia*
>
> https://en.wikipedia.org/wiki/Serverless_computing

- This is the source of truth, right? The place that we always turn to for an unbiased and unarguably correct point of view of the subject in question.
- In this case, the first line of the wikipedia page says, “Serverless computing is a cloud-computing execution model”,
- and agree that the compute aspect of serverless is a cloud based execution model, but the data storage aspect of serverless is not mentioned until later on, and is not given the same focus.

@cjrpriest

Utility Compute

Chris Priest

I think the name is misleading, I think a better / more descriptive name is "utility compute"

I want you to think for a second about typical utility services you have in your home

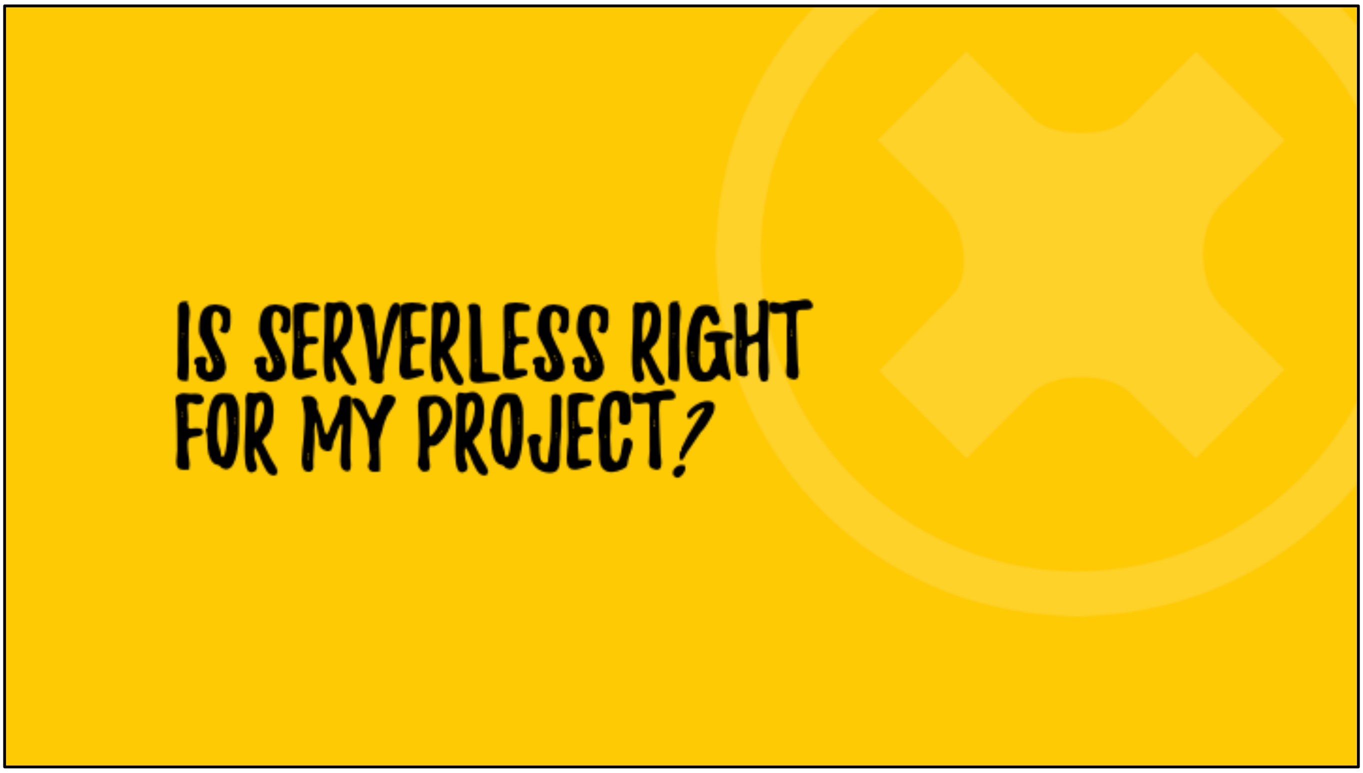
Water, gas, electricity

The question is, What makes them utility services

- You don't care about how the service gets to you (how it's generated or refined), or about the underlying infrastructure (how it's pumped or transmitted to you) — that is all outsourced to the energy company, water company
- They are on demand & made available at precisely the point that you need them
- They are metered — you pay for exactly what you use (more or less, there might be a small service charge)

This is much like how serverless operates.

You don't worry about underlying infrastructure
Available on-demand at precisely the point it's needed
You pay for what you use



IS SERVERLESS RIGHT FOR MY PROJECT?

So you have an understanding of what serverless is, how do you know if it’s right for your project?

Serverless / Instance-based Value Face Off

https://flic.kr/p/7UcCYp

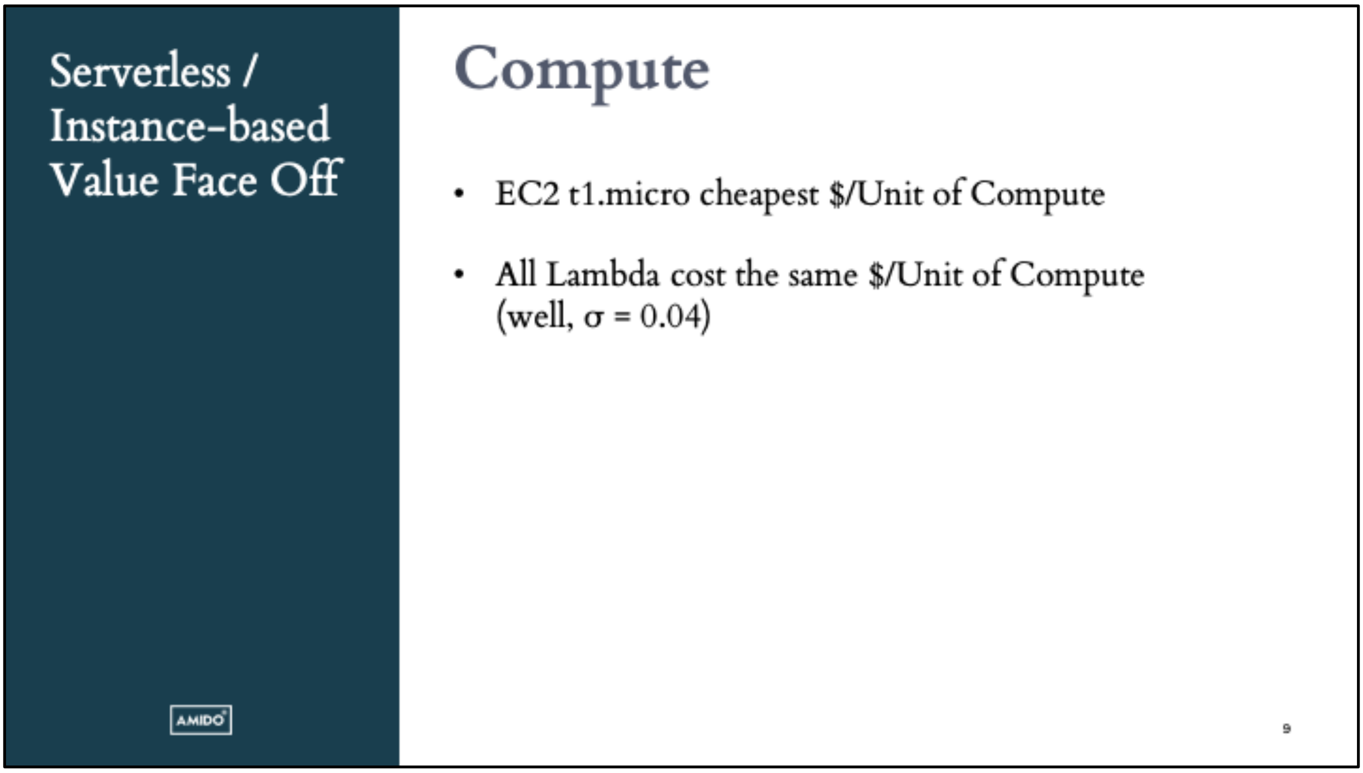
- I thought it would be interesting to do some analysis on exactly what you get, per $,
- when using instance-based vs serverless services on AWS

So for compute:

- Wanted to test and compare EC2 v Lambda
- for my compute tests I wrote a small bit of code in c#
  - Calculates large factorials / calculates large prime numbers
  - how many iterations of the predefined task can the target platform complete in a predefined period of time
  - I've called these packages of work "Units of Compute"
    - The more Units of Compute that can be processed in a specific amount of time, the more compute is available
  - e.g. for a particular environment, you might find that it's possible to run x number of iterations of a specified task in a period of 10 minutes, but that in another environment it's possible to run 2x iterations of the same task in the same time period (10 minutes)
    - by understanding the cost to run that compute environment for 10 minutes, we should be able to determine the relative cost of arbitrary compute.

  - My tests covered a number of EC2 instance types, and a number of different lambda memory configurations – remembering that with lambda the amount of CPU Amazon intends to assign to your function is directly proportional to the amount of memory you allocate

- For storage:
  - I created a large number of documents, containing random data, and stored them in various RDS configurations, and in S3
  - I tried both as key/value stores – i.e. no complex queries
  - I then (from within the AWS network) test response times & throughput that could be achieved from these services



Serverless / Instance-based Value Face Off

Compute

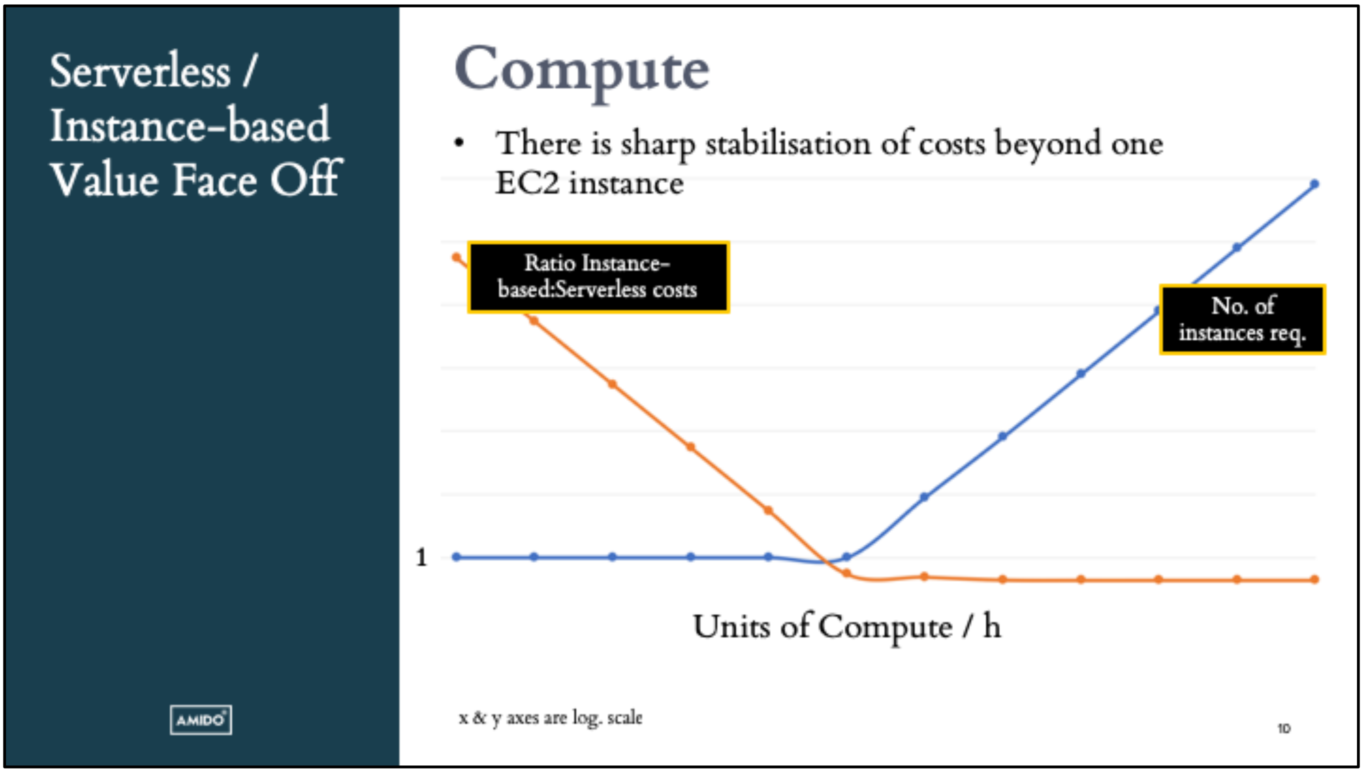
- EC2 t1.micro cheapest $/Unit of Compute
- All Lambda cost the same $/Unit of Compute (well, $\sigma = 0.04$)

9

- So what did I find? On the compute side:

- t1.micro – didn't test every possible instance type, but this one stood out
    - The hypothesis that I was chasing was that larger instance types would represent a lower $/Unit of Compute cost
    - However, it appears that the smallest instance types are those for which you get the most Unit of Compute per $

- Lambda costs are same
    - AWS got this right, there appears to be no compute advantage by using one configuration over another
    - i.e. when choosing more CPU for your lambda, your stuff gets done proportionally faster.
    - Why would you choose lower configs? If you need to wait for i/o – lambda CPU is idle
    - If your lambdas CPU bound, max out memory – you'll almost certainly be better off (faster response for no extra cost)



Serverless / Instance-based Value Face Off

Compute

- There is sharp stabilisation of costs beyond one EC2 instance

x & y axes are log. scale

10

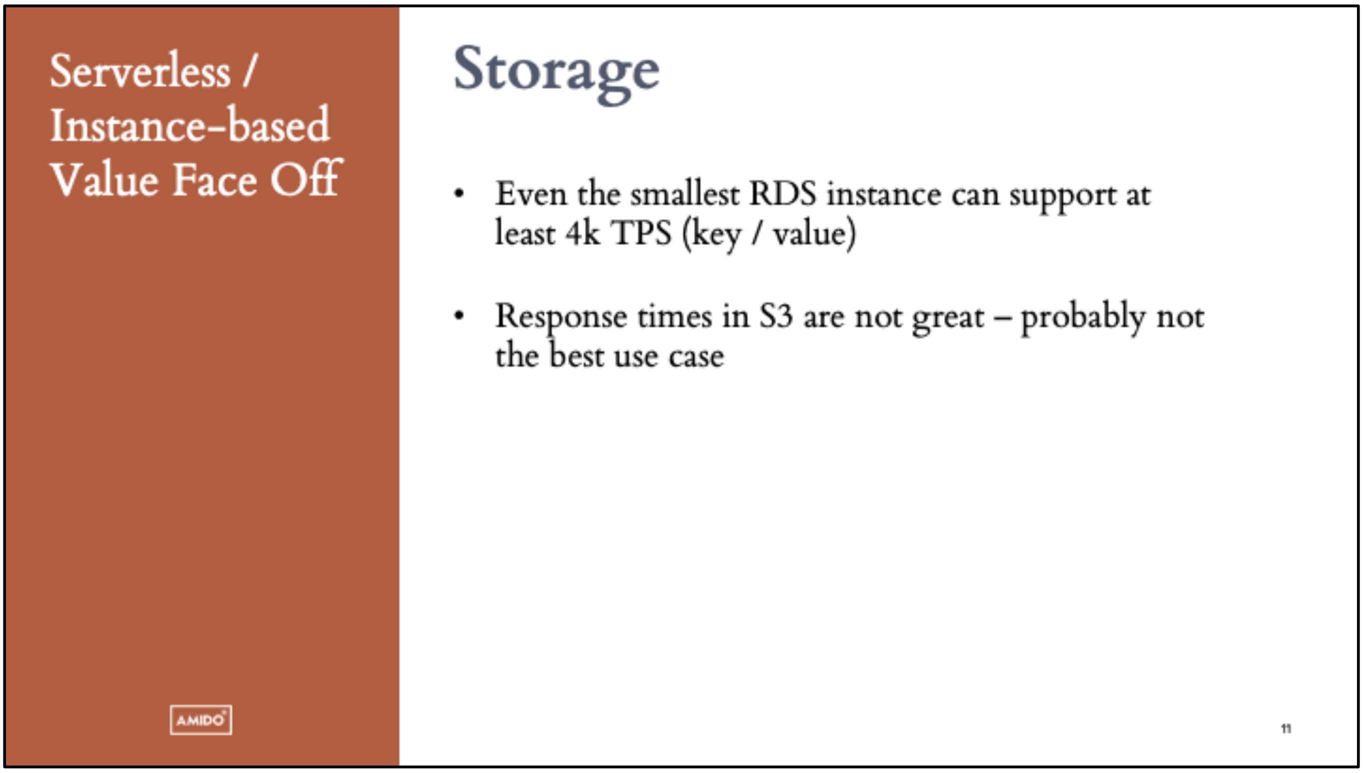
- Describe graph

- Pure compute / variable costs – revisit this later

- Lastly, here we are looking at the details of the cost of compute as the number of Units of Compute /h increases
  - Lines
    - Orange line: ratio of instance-based to serverless costs – as the line goes down, this means it is cheaper instance-based
    - Blue line: no. of instances required to service that number of Units of Compute per h

- Just looking at compute costs (and nothing else)
  - The cost to perform the same compute task in EC2 is significantly higher (compared to Lambda) until you require more than one instance to complete the work required
  - At the point of inflection – 1 instance – the ratio stabilizes and instance-based is roughly half the price of serveless compute (for a given task)

- In other words, unless you can keep at least one instance busy, it’s likely (in terms of pure compute costs) that it’ll be cheaper on a serverless platform
  - But there is more to this than pure compute / variable costs
  - Lets revisit this shortly



Serverless / Instance-based Value Face Off

Storage

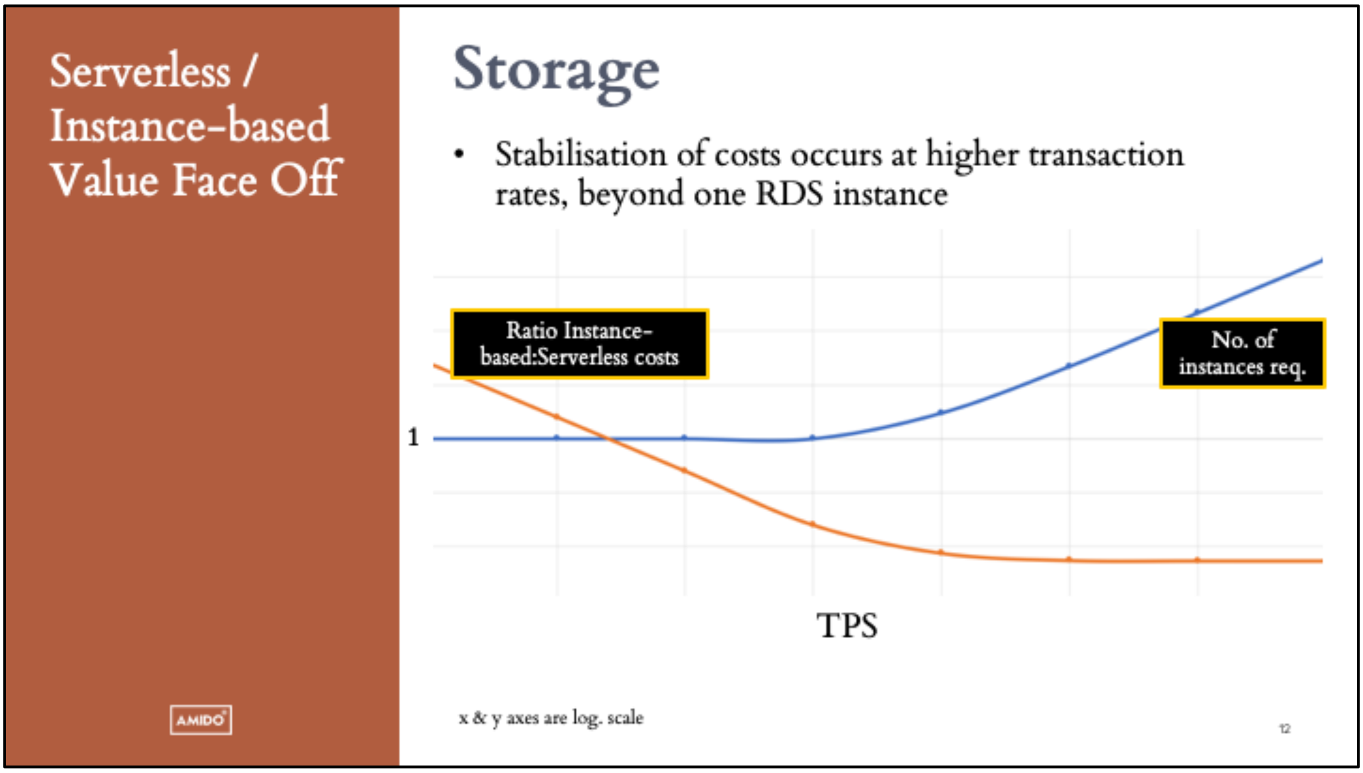
- Even the smallest RDS instance can support at least 4k TPS (key / value)
- Response times in S3 are not great – probably not the best use case

AMIDO

On the storage side, these are the significant things that I discovered

- First of all, that even the smallest RDS instances appear to have a very high level of performance when it came to retrieving values by key
  - In my tests I was able to sustain over 4k TPS with a db.t2.micro (the smallest available current generation db instance)
  - This surprised me – it immediately actually makes S3 & Dynamo look extremely expensive (when looking at the problem purely from a $/per request basis)

- Secondly – the response times for S3 were relatively poor compared to RDS. These differences (20-30X) may not actually be a problem for your use case – sometimes a response of 30ms is just as good as a response of 1ms
  - However, I recognize that DynamoDB (on-demand) is probably missing from these tests
  - I guess it's worth saying that S3 is going to work best in certain storage & client requirements, in particular where the clients are distributed



Serverless / Instance-based Value Face Off

Storage

- Stabilisation of costs occurs at higher transaction rates, beyond one RDS instance

x & y axes are log. scale

- Lastly, here we are looking at the details of the cost of serving a data retrieval request, as Transactions Per Second (TPS) increased
  - Lines
    - Orange line: ratio of instance-based to serverless costs – as the line goes down, this means it is cheaper instance-based
    - Blue line: no. of storage instances required to service those TPS

- We might expect that instance-based storage becomes cheaper after one instance – similar to compute
- What in fact we see, is that in general it becomes cheaper to service a given number of TPS with instance-based storage earlier than we might expect
- This correlated with the earlier observation that RDS (at least at a key value store), can look like pretty good value for money, if you are able to utilize an instace…

- Maybe serverless storage options aren't as mature as the compute side?

- It becomes cheaper to run an RDS instance (vs S3 storage) much earlier, under lower TPS load than I expected

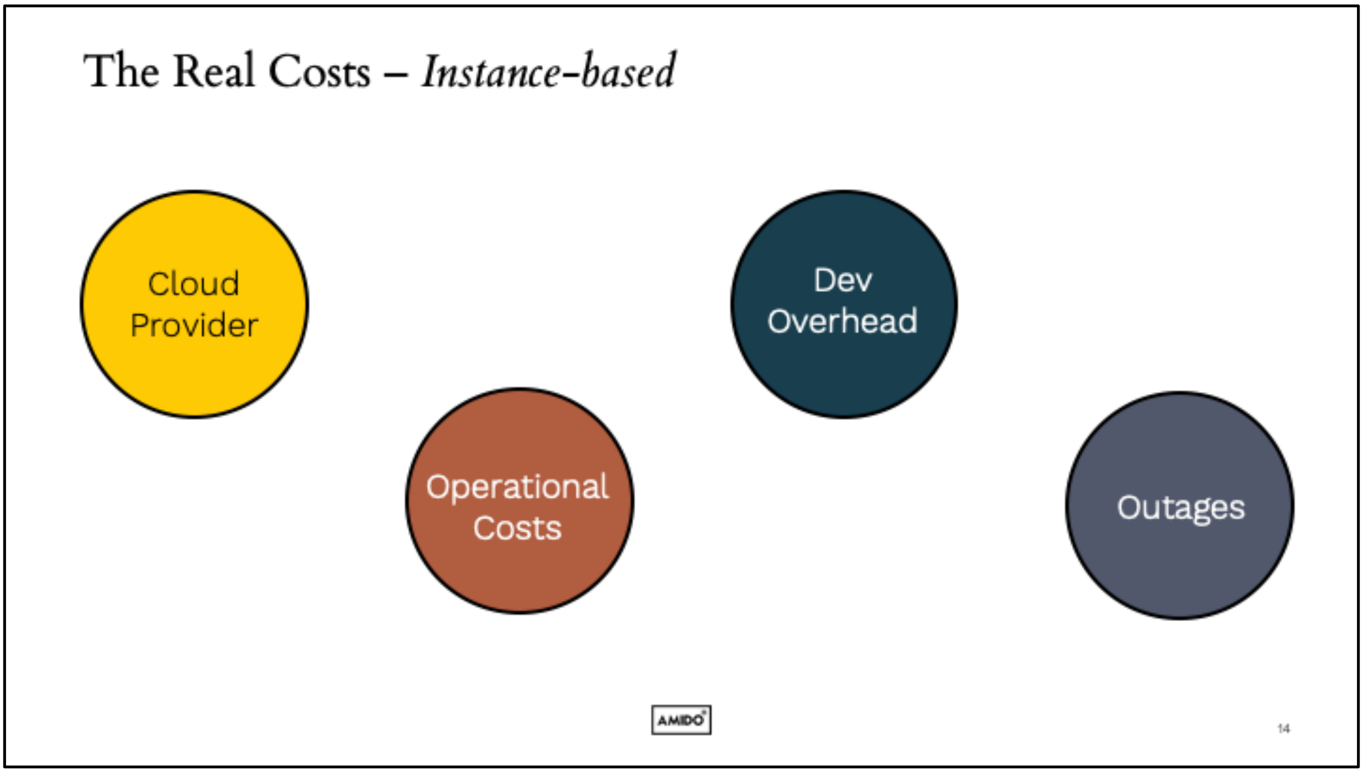
BUT WHAT ARE THE REAL COSTS?

So this is all interesting, but actually it's all a bit wrong

If there is one thing I want you to take away from this talk, this is it: if anyone presents to you direct cost-based justifications like these, for the purpose of avoiding serverless, think really carefully

They may be right – especially in very high load scenarios. But in the majority of cases it doesn't make sense

And heres why



The Real Costs – *Instance-based*

Cloud Provider

Operational Costs

Dev Overhead

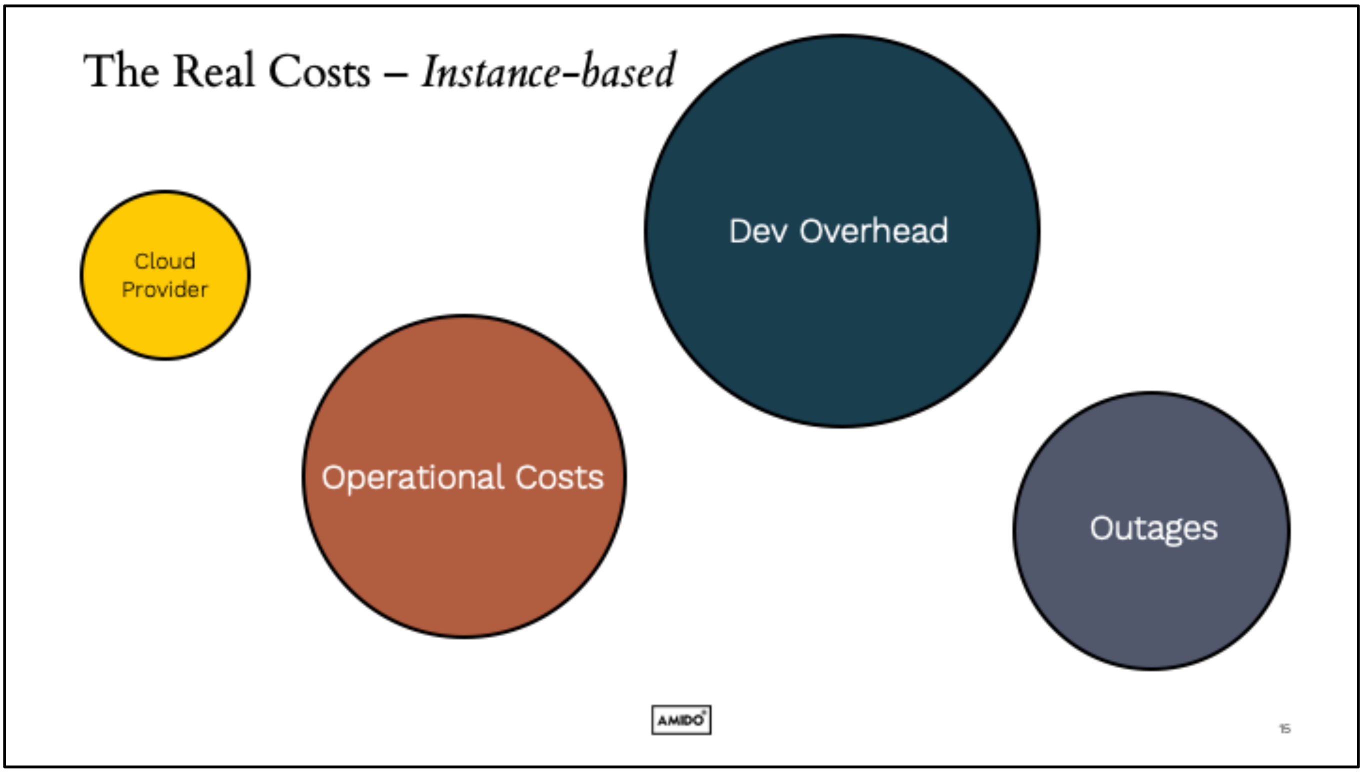
Outages

14

- Its very easy to get drawn into discussions around the direct / variable ($s paid to cloud provider) costs of compute & storage in the serverless landscape

- But what about the indirect costs?

- Just some of those indirect costs might be
    - Operational costs
        - IaaS monitoring — CPU, disk space, memory usage — all needs to be surfaced & reported with a reaction plan
        - Testing your application with new OS versions
            - Managing the rollout of a new OS version with no downtime
        - understanding, implementing & testing the scaling plan — which is no trivial task
    - Outages due to infrastructure – mismanagement
    - Dev overheard – this one is really interesting, I'll talk about this shortly



The Real Costs – *Instance-based*

Cloud Provider

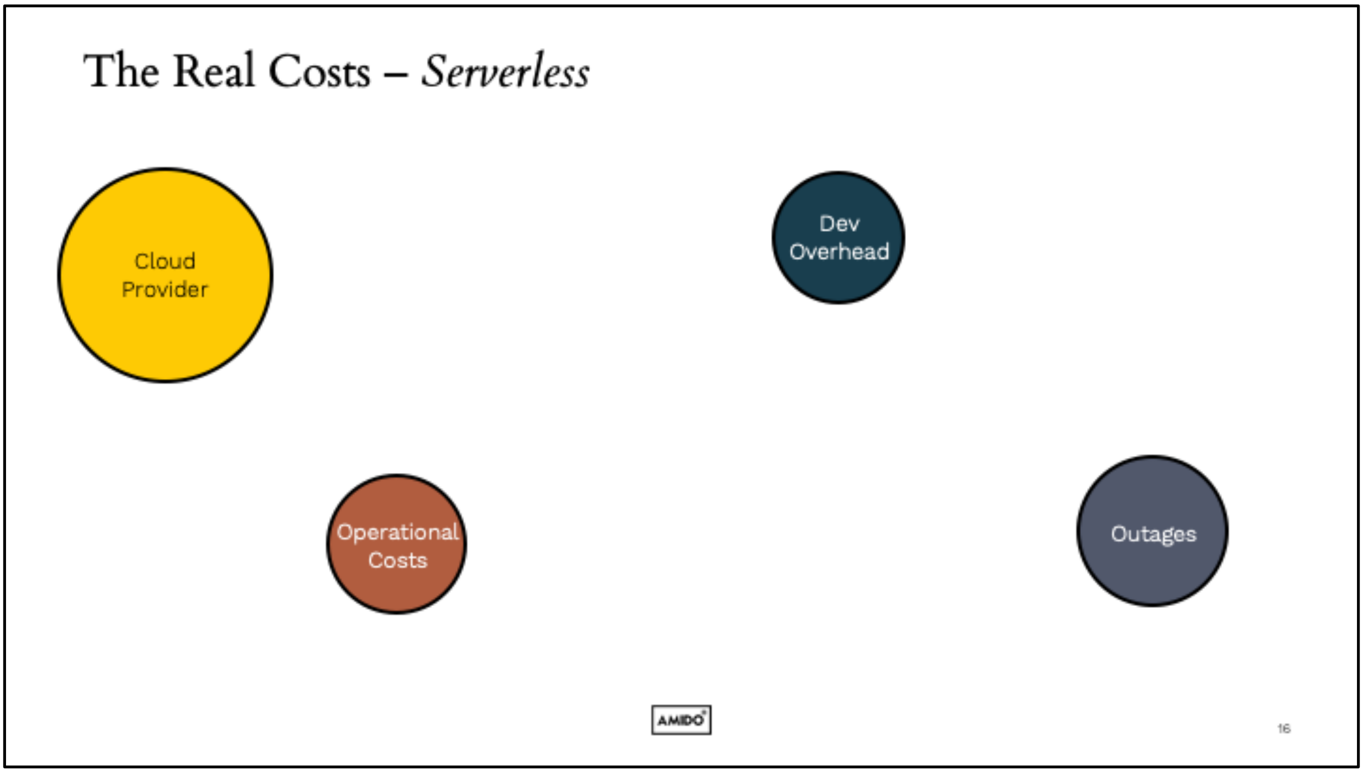
Operational Costs

Dev Overhead

Outages

- In terms of relative size -- i don’t know, I’m just guessing here, everyone’s situation will be different

- But what is likely is that your instance-based cloud provider costs are probably a relatively small part of your overall service provision costs

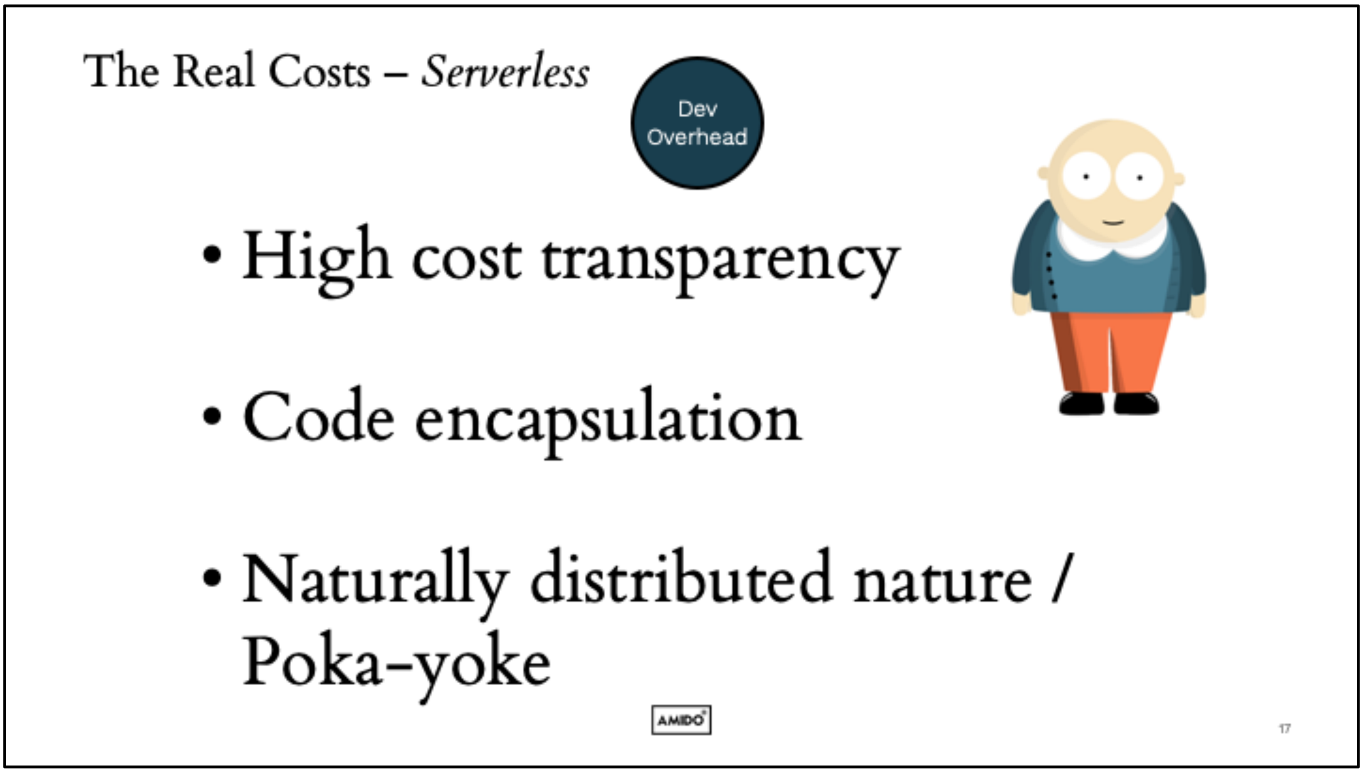


The Real Costs – *Serverless*

- If you move to a serverless way of operating, then you might have higher cloud provider costs (or they might be lower), but your other costs are likely to be much lower

- Operational costs – fairly obvious – if you don’t have servers & infrastructure to be concerned about, then the costs in this area reduce
- Outages – the cloud provider is probably better at looking after / maintaining infrastructure failure than you are

- So lets talk about dev overhead – I think this is where some of the largest, and often overseen cost savings can be made when moving to the serverless space



The Real Costs – *Serverless*

Dev Overhead

- High cost transparency
- Code encapsulation
- Naturally distributed nature / Poka-yoke

AMIDO

17

- So how is dev overhead reduced when it comes to serverless?

- High cost transparency: when the cost of running a service is highly transparent. The cost of running a service is transparent to the business, it becomes easier to prioritise your backlog inline with business requirements.
  - For example — in a non-serverless world, the team might be tempted to run a story around optimising some code that is highly inefficient.
  - It is not to say that the code is not highly inefficient, but if the performance to the end user is acceptable, and the absolute cost is low (very easy to measure, just filter your cloud costs on the function in question) then it's probably not worth doing that work
  - Avoiding waste / avoiding work that is not required is one of the most effectively cost savers out there

- Code encapsulation: this is more a benefit of FaaS, which usually comes in a serverless flavour — but if you've got some bad code, this could be encapsulated in a function.
  - If you don't need to change or maintain or update it, then this is effort

- Naturally distributed nature
    - In a world where most applications are a distributed nature, and probably event driven, a serverless execution model fits really well
    - In a way, this is an application of Poka-Yoke – a Japanese term for mistake-proofing – the prevention of unintended errors
    - If you fit the execution model to the design of the application, you are much less likely to make long terms (costly mistakes)

- So next time you are about to start a new project, remember bob – maybe you’ll want to go fieldless too

Chris Priest

@cjrpriest

Thank You

- I hope this gave you some alternative ideas on how to think about serverless

- If you've got any questions or comments, please send me a tweet, I'd love to hear them!.

- Thanks!