Serverless… on your own servers?

Evan Anderson, Google
Kubecon, Serverless Practitioner's Summit, 2019
Today, you will

Understand what Serverless is...

Learn how to see if your service is serverless

... and how to make it more so and delight your users.
Serverless is the worst name

Defining something by what it is not doesn't make a lot of sense.

And… it's not even true!

Your code has to run somewhere… there are still servers that run your code.

All your code runs on servers!
What does Serverless mean?

Three definitions:

- A development model where developers focus on a single unit of work. *(We're going to focus on this one.)*
- A platform which automatically scales without developer intervention. *(Enabled by the development model.)*
- Pay for what you use (granular billing by action).
Single Unit of Work?
Function as a Service

Unit of work:

“run this function once”

Developers focus on the business logic

Only one responsibility

Everything needed by the function is provided via configuration
Build as a Service

Unit of work:

“Run this build command once”

Examples:

- TravisCI
- CircleCI
- Google Cloud Build
Google BigQuery

Unit of work:

“Run this SQL query once”

What about MySQL, Postgres, etc?

• Large data sets require sharding
• Developers have to think about sharding when using the service!
Object Storage

Unit of work:

“Store this blob”

Lots of implementations…

• S3
• GCS
• Ceph
Why focus on unit of work?
Which do we care more about?

1. Developer productivity
2. Cost optimization

For most applications, the primary cost will be developer productivity.

Pay per-use is handy and makes sense as an application gets bigger (fortunately, serverless helps with resource pooling).
Reducing human cost

Standardized units of work let you scale expensive humans farther!

How many humans to manage EC2? 2000?

If Amazon has 1M machines, then the human cost per machine is $400/year

You won’t get AWS scale, but ops maintenance could be 1 engineer / 200 apps with serverless.

1M / 2000 * salary = 1/500th of a human’s salary ($200k?) = $400/year

Warning! Back of the envelope math!
Distractions

There's a lot of research that shows that humans are bad at multi-tasking, so "focus on one thing" makes a lot of sense.

What's standing in the way?

• Scaling: capacity / replication / sharding
• OS / library patching
• Testing (!)
• Rollout
• Monitoring
How do we get there?
Understand the data flow

Understand how developers use your service.

If you are providing general computing, provide scaffolding and rules of the road.

If you store data, is it read/write for computers or reporting for humans? Reflect this in your API.

If you are glue, your goal is to stick things together and get out of the way!
Multitenancy helps

Multitenancy isn’t required to have a serverless model. But, all the following are more serverless:

• Services over the network
• Provision a new domain (namespace) without adding resources
• Transparently scale backend resources without client updates

Multitenant isolation can be hard — it’s okay to fake it.
Managing Serverless

What are the common tasks developers need to visit your control plane? Make those tasks easy.

For Knative, we had 4 user journeys:

- First-time deploy
- Roll out new code or config
- Roll back during outage
- Canary new code or config

All of these take one control plane API call.
Kafka as Example

Unit of work:

“Push/Pop a persistent FIFO”

Partitions don't need to map 1:1 to servers.

Micro-partitions allow underlying capacity to change without involving developers.

Topics allow different teams to share capacity.

1000 partitions per topic
What’s not serverless (today)?

- Limits on total partitions per cluster across all topics \((N \times \text{brokers (servers)} \times \text{replication})\).
- Failover semantics depend on individual nodes (coordinator).
- “Cluster address” is… a list of bootstrap servers?
Antipatterns

Serverless only works if you understand how your problem breaks down into independent units of work. These are less serverless:

- Developers scale out in server units
- Client code needs to know individual servers
- Stickiness and server-side state
- Per-cluster rather than per-domain settings

...if they matter to developers
Living the Serverless Dream?

If you’re on a Platforms Team:
  • Push developers toward scale-out platforms

If you’re writing Infrastructure Software:
  • Make it easy for teams to share one install

If you’re building applications:
  • Vote with your feet!