Towards Serverless as Commodity
a Case of Knative

WoSC 2019, Davis, California

Nima Kaviani, PhD
nkavian@us.ibm.com
@nimak

Dmitriy Kalinin
dkalinin@pivotal.io
@dmitriykalinin

Michael Maximilien
maxim@us.ibm.com
@maximilien

Serverless (+)

- is easier to manage
- is cost effective

Serverless (-)

very easily results in vendor lock-in

Serverless

“... is one of the worst forms of **proprietary lock-in** we've ever seen in the history of humanity”

Alex Polvi - CoreOS CEO

Reference: https://www.theregister.co.uk/2017/11/06/coreos_kubernetes_v_world/

Kelsey Hightower @kelseyhightower · Oct 23

I'm in the same boat regarding the fear of "lock-in". We are addressing this at GCP by backing our Serverless offerings with open source projects.

Fully managed platforms with an escape hatch for those that need it.

> 
> **kj** @dam · Oct 23
>
> Replying to @kelseyhightower
>
> That is a very fair point.
>
> The one contributing factor I hear the most is fear of "lock-in"
>
> Which IMHO means that the other clouds have been more resistant to changing their business than they could be.
>
> I would LOVE this to be a non-issue.

Kubernetes won the CaaS war

Question is:

who will win the Serverless war?

Kubernetes won the CaaS war

Kubernetes Success

from a ***Provider***'s perspective

- It is open source
- It is IaaS agnostic

Kubernetes Success

from an ***Operator***'s perspective

- Declarative operations
- Consistent deployment

Kubernetes Success

from a ***Developer***'s perspective

- Consistent API across vendors
- Makes migration easy

What would it take for a ***serverless*** platform to repeat ***Kubernetes***' success?

Kelsey Hightower
@kelseyhightower

Following

This is why Knative is important. Innovation in infrastructure becomes utility once interoperability and interchangeability are possible. It's not about rolling your own serverless stack, but having enough options so you don't have to. github.com/knative

> **John Arundel** @bitfield
> Lambda and serverless is one of the worst forms of proprietary lock-in that we've ever seen in the history of humanity. It's seriously as bad as it gets. You'll never be able to run your application without Amazon's infrastructure theregister.co.uk/2017/11/06/cor…

Learnings from Kubernetes

- Open source
- IaaS agnostic
- Consistent deployment model
- Consistent API across vendors

Learnings from Kubernetes

- Open source
- IaaS agnostic
- Consistent deployment model
- Consistent API across vendors

Consistent API Model

AWS Lambda
Apache OpenWhisk
OpenFaaS
Kubeless
Knative

1. Packaging Contract
2. Runtime Invocation Contract
3. Application Invocation Contract
4. Execution Model
5. Retry Model
6. Concurrency Model
7. Traffic Splitting

Excluding streaming scenarios or where an open connection to the service is required.

Packaging Contract

Platform	*Lambda*	*OpenWhisk*	*OpenFaaS*	*Kubeless*	*Knative*
	Custom Packaging	OCI Image + Custom Packaging	OCI Image + Custom Packaging into OCI Image	OCI Image + Custom Packaging into OCI Image	OCI Image

Runtime Invocation Contract

Definition:

The API boundary between the platform and the runtime

Platform	***Lambda*** λ	***OpenWhisk***	***OpenFaaS***	***Kubeless***	***Knative***
	HTTP Service (*pull based)* *Pull from Lambda API Runtime*	HTTP Service (*push based)* *Push to Application Runtime*	HTTP Service (*push based)* *Push to Watchdog*	HTTP Service (*push based)* *Push to Application Runtime*	None

Application Invocation Contract

Definition:

The API Boundary between the runtime & application in / out

Platform	*Lambda* λ	*OpenWhisk*	*OpenFaaS*	*Kubeless*	*Knative*
	JSON Envelope	JSON Envelope Opt-in HTTP/1	Stdin / Stdout	Stdin / Stdout	HTTP/1 HTTP/2 CloudEvents

Execution Model

Sync vs Async

Platform	*Lambda*	*OpenWhisk*	*OpenFaaS*	*Kubeless*	*Knative*
	Sync / Async Specify `InvocationType`	Sync / Async `Non-Blocking` Invocations Query with Invocation id	Sync / Async `Non-Blocking` Invocations (NATS) Callback for results	Sync / Async Pub/Sub trigger Support (Kafka / NATS)	Sync

Retry Model

Only done for async workload!

Platform	*Lambda* λ	*OpenWhisk*	*OpenFaaS*	*Kubeless*	*Knative*
	Functional Failures `DeadLetterQueue` for failures	None	On timeout	None	No aysnc workload ⇒ No retries

Concurrency Model & Autoscaling

Platform	*Lambda*	*OpenWhisk*	*OpenFaaS*	*Kubeless*	*Knative*
	Request-based Autoscaling by queue length	Request-based	Request-based Resource-based Uses Prometheus metrics to drive autoscaling	Request-based Resource-based Uses Kubernetes HPA No scale-to-zero	Request-based (KPA) Resource-based (HPA)

Traffic Splitting

Platform	*Lambda*	*OpenWhisk*	*OpenFaaS*	*Kubeless*	*Knative*
	Built-in First class app revision	External Load Balancing (e.g. nginx)	External Service-Mesh Istio / Linkerd	External Service-Mesh Istio / Linkerd	Built-in First class app revisions Managed Routing

What would be the ideal design for a *serverless* platform?

Packaging Contract

Discussion

1. ***OCI Images***
2. ***Custom Packaging***
3. ***Custom Packaging into OCI Image***

Packaging Contract

Discussion

1. ***OCI Images***
2. ***Custom Packaging***
3. ***Custom Packaging into OCI Image***

Runtime Invocation Contract

Discussion

1. ***Runtime calls Platform***
2. ***Platform calls Runtime***

Runtime Invocation Contract

Discussion

1. ***Runtime calls Platform***
2. ***Platform calls Runtime***

Application Invocation Contract

Discussion

1. ***Custom Msg. Envelope***
2. ***Stdin / Stdout***
3. ***HTTP***
4. ***HTTP + CloudEvent***

Application Invocation Contract

Discussion

1. *Custom Msg. Envelope*
2. *Stdin / Stdout*
3. *HTTP*
4. *HTTP + CloudEvent*

Execution Model

Discussion

1. ***Sync***
2. ***Async***
3. ***Both***

Execution Model

Discussion

1. *Sync*
2. *Async*
3. *Both*

Retry Model

Discussion

1. ***Platform provided***
2. ***Leave it to the client***

Retry Model

Discussion

1. ***Platform provided***
2. ***Leave it to the client***

Concurrency Model & Autoscaling

Discussion

1. ***Pull-based # Req***
2. ***Push-based # Req***
3. ***Resource-based***

Concurrency Model & Autoscaling

Discussion

1. ***Pull-based # Req***
2. ***Push-based # Req***
3. ***Resource-based***

Traffic Splitting

Discussion

1. ***Native App Revisions***
2. ***Independent Apps***

Traffic Splitting

Discussion

1. ***Native App Revisions***
2. ***Independent Apps***

Questions?