Towards Serverless as Commodity
a Case of Knative

WoSC 2019, Davis, California
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?
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](http://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...](http://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

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.

AWS Lambda
Apache OpenWhisk
OpenFaaS
Kubeless
Knative
# Packaging Contract

<table>
<thead>
<tr>
<th>Platform</th>
<th>Lambda</th>
<th>OpenWhisk</th>
<th>OpenFaaS</th>
<th>Kubeless</th>
<th>Knative</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Custom Packaging</td>
<td>OCI Image + Custom Packaging</td>
<td>OCI Image + Custom Packaging into OCI Image</td>
<td>OCI Image + Custom Packaging into OCI Image</td>
<td>OCI Image</td>
</tr>
</tbody>
</table>
## Runtime Invocation Contract

**Definition:**
The API boundary between the platform and the runtime

<table>
<thead>
<tr>
<th>Platform</th>
<th>Lambda</th>
<th>OpenWhisk</th>
<th>OpenFaaS</th>
<th>Kubeless</th>
<th>Knative</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>HTTP Service (pull based)</td>
<td>HTTP Service (push based)</td>
<td>HTTP Service (push based)</td>
<td>HTTP Service (push based)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pull from Lambda API Runtime</td>
<td>Push to Application Runtime</td>
<td>Push to Watchdog</td>
<td>Push to Application Runtime</td>
</tr>
</tbody>
</table>
**Application Invocation Contract**

**Definition:**
The API Boundary between the runtime & application in / out

<table>
<thead>
<tr>
<th>Platform</th>
<th>Lambda</th>
<th>OpenWhisk</th>
<th>OpenFaaS</th>
<th>Kubeless</th>
<th>Knative</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>JSON Envelope</td>
<td>JSON Envelope</td>
<td>Stdin / Stdout</td>
<td>Stdin / Stdout</td>
<td>HTTP/1 HTTP/2 CloudEvents</td>
</tr>
<tr>
<td></td>
<td>Opt-in HTTP/1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Execution Model

### Sync vs Async

<table>
<thead>
<tr>
<th>Platform</th>
<th>Lambda</th>
<th>OpenWhisk</th>
<th>OpenFaaS</th>
<th>Kubeless</th>
<th>Knative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync / Async Specify InvocationType</td>
<td>Sync / Async</td>
<td>Non-Blocking Invocations</td>
<td>Non-Blocking Invocations (NATS)</td>
<td>Sync / Async Pub/Sub trigger Support (Kafka / NATS)</td>
<td>Sync</td>
</tr>
<tr>
<td>Query with Invocation id</td>
<td>Callback for results</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Retry Model

*Only done for async workload!*

<table>
<thead>
<tr>
<th>Platform</th>
<th>Lambda</th>
<th>OpenWhisk</th>
<th>OpenFaaS</th>
<th>Kubeless</th>
<th>Knative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional Failures</td>
<td>None</td>
<td>On timeout</td>
<td>None</td>
<td>No async workload ⇒</td>
<td>No retries</td>
</tr>
<tr>
<td>DeadLetterQueue for failures</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Concurrency Model & Autoscaling

<table>
<thead>
<tr>
<th>Platform</th>
<th>Lambda</th>
<th>OpenWhisk</th>
<th>OpenFaaS</th>
<th>Kubeless</th>
<th>Knative</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Request-based</td>
<td>Request-based</td>
<td>Request-based Resource-based</td>
<td>Request-based Resource-based</td>
<td>Request-based (KPA)</td>
</tr>
<tr>
<td></td>
<td>Autoscaling by queue length</td>
<td>Uses Prometheus metrics to drive autoscaling</td>
<td>Uses Kubernetes HPA</td>
<td>No scale-to-zero</td>
<td>Resource-based (HPA)</td>
</tr>
</tbody>
</table>
# Traffic Splitting

<table>
<thead>
<tr>
<th>Platform</th>
<th>Lambda</th>
<th>OpenWhisk</th>
<th>OpenFaaS</th>
<th>Kubeless</th>
<th>Knative</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Built-in</td>
<td>External Load Balancing (e.g. nginx)</td>
<td>External Service-Mesh Istio / Linkerd</td>
<td>External Service-Mesh Istio / Linkerd</td>
<td>Built-in First class app revisions Managed Routing</td>
</tr>
</tbody>
</table>
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
1. Runtime calls Platform
2. Platform calls Runtime
1. Runtime calls Platform
2. Platform calls Runtime

Discussion
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?