Writing a K8s Operator for Knative Functions

DevConf.CZ 2023

Luis Tomas Bolivar
Jose Castillo Lema
Agenda

1. Introduction
   a. PHYSICS project

2. Serverless / Function as a Service (FaaS)
   a. Knative

3. Operator pattern

4. Hands on
   a. Lab description
   b. Prizes
   c. Useful links
whoami

https://ltomasbo.wordpress.com/

https://www.linkedin.com/in/luis-tomas-bolivar-a1022

260/

https://github.com/luis5tb

ltomasbo@redhat.com
whoami

https://josecastillolema.github.io/

https://www.linkedin.com/in/jose-castillo-LEMA

https://github.com/josecastillolema

jlema@redhat.com
Introduction
Research Interest Groups (RIGs)

- EMEA
- Israel
**Project Goals**

Visual programming environment to create serverless workflows with reusable patterns and increased semantics

Platform-level functionalities to orchestrate and deploy FaaS workflows and optimize cloud/edge interplay

Provider-local resource management mechanisms to offer competitive and optimized services execution

This project has received funding from the European Union’s horizon 2020 research and innovation programme under grant agreement no 101017047
Challenges targeted by PHYSICS

- Abstract usage of **service offerings and clusters across the Continuum**
- Adaptation of code to new **serverless** computing paradigms
- Investigation of **space** (location of execution)-**time** (duration of execution) in the continuum
- Optimization of resource **selection** and operation (**global** and **local** level)
- Multiple Exploitation Channels and **Reusable** Artefacts
OPtimized HYbrid Space-time serviCe Continuum in faaS

Architecture / Baseline technologies

This project has received funding from the European Union’s horizon 2020 research and innovation programme under grant agreement no 101017047
The talk will focus on a design and development environment coming from the H2020 PHYSICS project, that aims to ease application evolution to the new FaaS model. It uses the Node-RED open source tool as the main function and workflow runtime. The goal of the environment is to enable a more user friendly and abstract function and workflow creation process for complex FaaS applications. To this end, it provides an extendable, pattern-enriched palette of ready-made, reusable functionalities such as workload parallelization, data collection at the edge, function orchestration creation among others. The environment embeds seamless DevOps processes for generating the deployable artefacts of the FaaS platform (Openwhisk). Annotation mechanisms are also available for the developer to dictate diverse execution options towards the deployment stacks, including sizing and locality considerations, as well as abilities for dynamic FaaS applications to continuously leverage the edge-cloud continuum.
Serverless has been gaining popularity as a new way to program and deploy applications on clouds. Function as a service (FaaS) is an approach encompassed by serverless, extending the FaaS concept by avoiding server infrastructure management.

In this context, functions rely on containers, and deploying new containers can cause several overheads to the platforms and the function's execution (cold start delays).

Kubernetes-based platforms are used for serverless proposes, and K8S provides an ImageLocality mechanism to address it, but it relies on entire warm containers and not on layers.

Therefore, we propose and implement on K8S two new scheduling policies. The first is a ContainerLayer-Aware policy that optimizes function's placements by selecting machines with the biggest rate of container layers that can be shared. The second is a Multi-Objective policy for heterogeneous platforms that reduces at the same time the makespan and the data transferred by functions I/O and container layers.
Serverless architectures

Removes the need for….

- Provisioning and Utilization
- Operations and Management
- Scaling
- Availability and Fault Tolerance
The Core Serverless Traits

- Easy to get started
- Event Driven
- Stateless
- Distributed and Elastic
- Scale on demand
FaaS in the cloud

- Amazon Lambda
- Azure Functions
- IBM Cloud Functions
- Google Cloud Functions
FaaS in the K8s
Knative
Bringing Serverless Applications to Kubernetes

**SERVING**

A *request-driven* model that serves the *container* with your application and can "scale to zero".

**EVENTING**

Common *infrastructure* for consuming and producing *events* that will stimulate applications.

**FUNCTIONS**

A *programming* model that lets you focus on *just your code* for faster iterations.

**CLIENT (kn)**

Allows you to create resources interactively from the *command line* or from within scripts.
The "Serverless Pattern"

Event | trigger | Your Application | produce | Results

HTTP Requests
Kafka Messages
Image Uploaded
New Order
Login from user
The "Serverless Pattern"

A serverless web application

- Browser
  - HTTP Request: `myapp.example.com`
- Container
- Database

- Browser
  - HTTP Requests
- Container
The "Serverless Pattern"

Processing a Kafka message

External System -> Kafka Message -> Container -> Storage

External System -> Kafka Messages -> Container

Processing a Kafka message
The "Serverless Pattern"

A serverless web application

Benefits of this model:

- No need to setup auto-scaling and load balancers
  - Scale down and save resources when needed.
  - Scale up to meet the demand.
- No tickets to configure SSL for applications
- Enable Event Driven Architectures (EDA) patterns
- Enable teams to associate cost with IT
- Modernize existing applications to run as serverless containers
Where Serverless?

- Application with unpredictable or bursty number of requests.
- Maximum resource utilization to reduce the carbon footprint
- Building event-driven, loosely coupled systems
- Low barrier for Developers (Kubernetes is hard)
- A/B testing or canary deployments
- Seasonal or periodic workload.
- microservices or containers and want to leverage serverless
Cashless payment systems
Transaction processing auditing
Fraud Detection
Credit checks
Check signature validation through OCR

Product thumbnail generation
Chatbots and CRM functions
Marketing Campaign notifications
Sales Audit
Content Push

Image results validation (X-rays, MRIs)
Fast Healthcare Interoperability Resources
Queries
Result notifications
Scheduling services
Test result requests (PDFs, Reports)

Network Anomaly detection (VNF)
Victim Identification
Network Feature enablement
Traffic Manipulation
Media processing (5G and VNFs)
Serverless Operational Benefits

**Over provisioning**
- Time in capacity planning
- IT cost of idle resources

**Under provisioning**
- Lost business revenue
- Poor quality of service

**More applications**
- Direct line between IT costs & business revenue

**NOT Serverless**

**with Serverless**
Installation experience

"Easy day 1 and even better for day 2"

➤ Click Install experience
➤ Developer & admin experience in Console
➤ Built-in event sources
➤ No external dependencies.
➤ "Just works."
Simplify application development/deployment on K8

Reduce developer toil and cognitive overhead with Knative tools
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
labels:
  app: guestbook
spec:
  replicas: 1
template:
  metadata:
    labels:
      app: guestbook
tier: frontend
spec:
  containers:
  - image: markusthoemmes/guestbook
    name: guestbook
tier: frontend
    resources:
      requests:
        cpu: 100m
        memory: 100Mi
    env:
      - name: GET_HOSTS_FROM
        value: dns
    ports:
    - containerPort: 80

apiVersion: v1
kind: Service
metadata:
  name: frontend-service
tier: frontend
spec:
  ports:
  - port: 80
    selector:
      app: guestbook
      tier: frontend
---
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: frontend-route
spec:
  to:
    kind: Service
    name: frontend-service

apiVersion: extensions/v1beta1
kind: HorizontalPodAutoscaler
metadata:
  name: guestbook
  namespace: default
spec:
  scaleRef:
    kind: ReplicationController
    name: guestbook
    namespace: default
    subresource: scale
  minReplicas: 1
  maxReplicas: 10
  cpuUtilization:
    targetPercentage: 50

apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: frontend
spec:
template:
  metadata:
    labels:
      app: guestbook
tier: frontend
spec:
  containers:
  - image: markusthoemmes/guestbook
tier: frontend
    resources:
      requests:
        cpu: 100m
        memory: 100Mi
    env:
      - name: GET_HOSTS_FROM
        value: dns
    ports:
    - containerPort: 80
---
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: frontend-route
spec:
  to:
    kind: Service
    name: frontend-service

---
apiVersion: v1
kind: Service
metadata:
  name: frontend-service
tier: frontend
spec:
  ports:
  - port: 80
    selector:
      app: guestbook
      tier: frontend
---
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: frontend-route
spec:
  to:
    kind: Service
    name: frontend-service
Knative Functions

**Powerful Developer experience**
- Local Developer Experience
- IDE Developer Experience
- Offers multiple build strategies
- Deploy as Knative Service
- Project templates
- Support for Cloud Events/HTTP
- On cluster build using Tekton/Pipelines
- **Runtimes:**
  - Node
  - Go
  - R
  - Python
  - TypeScript
Operator pattern
Example: Kafka operator

```yaml
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  labels:
    app: my-cluster
    name: my-cluster
    namespace: myproject
spec:
  # ...
  kafka:
    replicas: 3
    # ...
```
Operator pattern
Operators simplify management of complex applications on Kubernetes

- Encode human operational knowledge
- Automatically patch, upgrade, recover, and tune container-based apps and services
- Kubernetes-native
- Purpose-built for a specific application or service
- Enable “day 2” management
Encoding and automating Ops knowledge

WITHOUT OPERATORS: REACTIVE
- Continually checks for anomalies
- Alert humans for response
- Requires manual change to fix

WITH OPERATORS: PROACTIVE
- Continually adjusts to optimal state
- Automatically acts in milliseconds

Red Hat Operator Development solution
The Operator Framework is an open source toolkit to build and manage Kubernetes Operators, in an effective, automated, and scalable way.

For Builders and the community
- Easily create application on Kubernetes via a common method
- Provide standardized set of tools to build consistent apps

For application consumers and Kubernetes users
- Keep installed apps up to date for security reasons and app lifecycle management
- Consume of cloud-native / kube-native applications more secure and easier
Choosing the right tool

**HELM**
- Implementation is declarative and simple
- Operator functionality is limited to Helm features
- Operator manifest bundle files (CRD, RBAC, Operator Deployment) are automatically generated

**ANSIBLE**
- Implementation is declarative and human-readable
- Ansible can express almost any operator functionality
- Operator manifest bundle files (CRD, RBAC, Operator Deployment) are automatically generated

**GO**
- Implementation is imperative and more complex
- There is no limit on the functionality you want to implement
- Operator manifest bundle files (CRD, RBAC, Operator Deployment) are generated from the Go source code
Hands on
Lab description

Goal:

- Create a golang operator that deploys an existing knative function
  - The Knative function has previously been built and pushed to a private registry
- Extend the operator to build, push and deploy a function located on a github repository

Example CRD:

```yaml
apiVersion: knf.example.com/v1alpha1
kind: KnativeFunction
metadata:
  name: knativefunction-sample
spec:
  name: test-function
  image: localhost:50000/kn-user/test-hw@sha256:79c456
  maxscale: "2"
  minscale: "1"
  concurrency: 1
```

Example output:

```
$ kubectl get knativefunction
NAME      AGE
knativefunction-sample   5m8s

$ kubectl get ksvc
NAME     URL
test-function http://test-function.default.127.0.0.1.sslip.io

$ curl http://test-function.default.127.0.0.1.sslip.io
DevConf.cz 2023!
```
Prizes

3 Logic | Robotárna boards - [github](https://github.com/logic-sz)

- Universal programmable toy designed for teaching programming
- Resembles a game console
- It has 100 RGB LEDs that can serve as a display, it has numerous buttons and a buzzer
- Powered by ESP32 microcontroller
- Kids can create custom games and learn programming while doing so
- Possible to run multiplayer games as the on-board processor features both, WiFi and Bluetooth 4
Useful links

- Knative documentation: [https://knative.dev/docs/](https://knative.dev/docs/)
- Knative.dev/client Golang API: [https://pkg.go.dev/knative.dev/client](https://pkg.go.dev/knative.dev/client)
- Operator SDK - Go Operator tutorial: [https://sdk.operatorframework.io/docs/building-operators/golang/tutorial/](https://sdk.operatorframework.io/docs/building-operators/golang/tutorial/)
- Intermediate Kubernetes Operators on IBM Developer Skills Network: [https://courses.course-dev.skills.network/courses/course-v1:IBMSkillsNetwork+CO0201EN+2021T1/course/](https://courses.course-dev.skills.network/courses/course-v1:IBMSkillsNetwork+CO0201EN+2021T1/course/)
Thank you

Red Hat is the world’s leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.

linkedin.com/company/red-hat
youtube.com/user/RedHatVideos
facebook.com/redhatinc
twitter.com/RedHat