




# Cloud Native with Serverless

Kevin Dubois

Sr Solution Architect

  @kevindubois



```
1 package com.redhat.developer.demo;
2
3 import javax.ws.rs.GET;
4 import javax.ws.rs.Path;
5
6 import io.vertx.core.logging.Logger;
7 import io.vertx.core.logging.LoggerFactory;
8
9 @Path("/businessvalue")
10 public class BusinessValue {
11
12     private final String BUSINESS_VALUE =
13         System.getenv().getOrDefault("BUSINESS_VALUE", "Customer happiness");
14
15     private final Logger log = LoggerFactory.getLogger(BusinessValue.class);
16
17     @GET
18     @Path("/")
19     public String generateValueToMyBusiness() {
20         log.info("Generating " + BUSINESS_VALUE);
21
22         // do something that brings value to the business
23
24         return BUSINESS_VALUE + " has been generated";
25     }
26 }
```

```
1 package com.redhat.developer.demo;
2
3 import javax.ws.rs.GET;
4 import javax.ws.rs.Path;
5
6 import io.vertx.core.logging.Logger;
7 import io.vertx.core.logging.LoggerFactory;
8
9 @Path("/businessvalue")
10 public class BusinessValue {
11
12     private final String BUSINESS_VALUE =
13         System.getenv().getOrDefault("BUSINESS_VALUE", "Customer happiness");
14
15     private final Logger log = LoggerFactory.getLogger(BusinessValue.class);
16
17     @GET
18     @Path("/")
19     public String generateValueToMyBusiness() {
20         log.info("Generating " + BUSINESS_VALUE);
21
22         // do something that brings value to the business
23
24         return BUSINESS_VALUE + " has been generated";
25     }
26 }
```



93

91

89

87

85

83

81

79

77

75

73

71

69

67

65

63

61

59

57

55

53

51

49

47

45

43

41

39

37

35

33

31

29

27

25

23

21

19

17

15

13

11

9

7

5

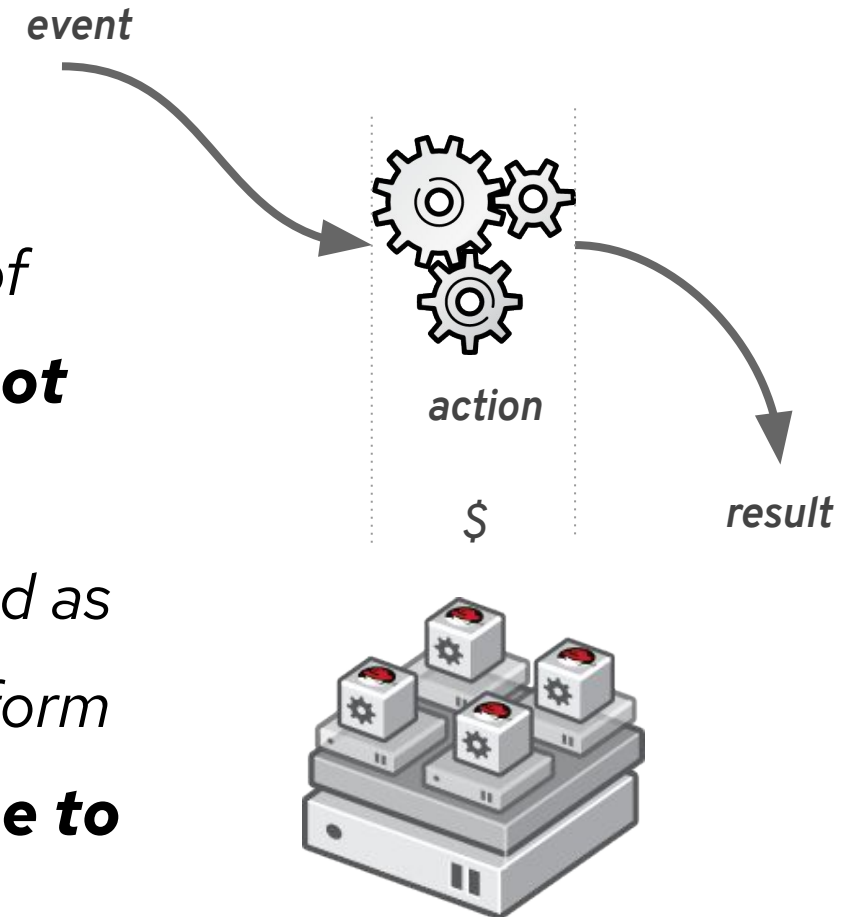
3

1



# Serverless Defined

*"Serverless computing refers to the concept of building and **running applications that do not require server management**. It describes a deployment model where applications, bundled as one or more functions, are uploaded to a platform and **executed, scaled and billed in response to the exact demand at the moment**"*

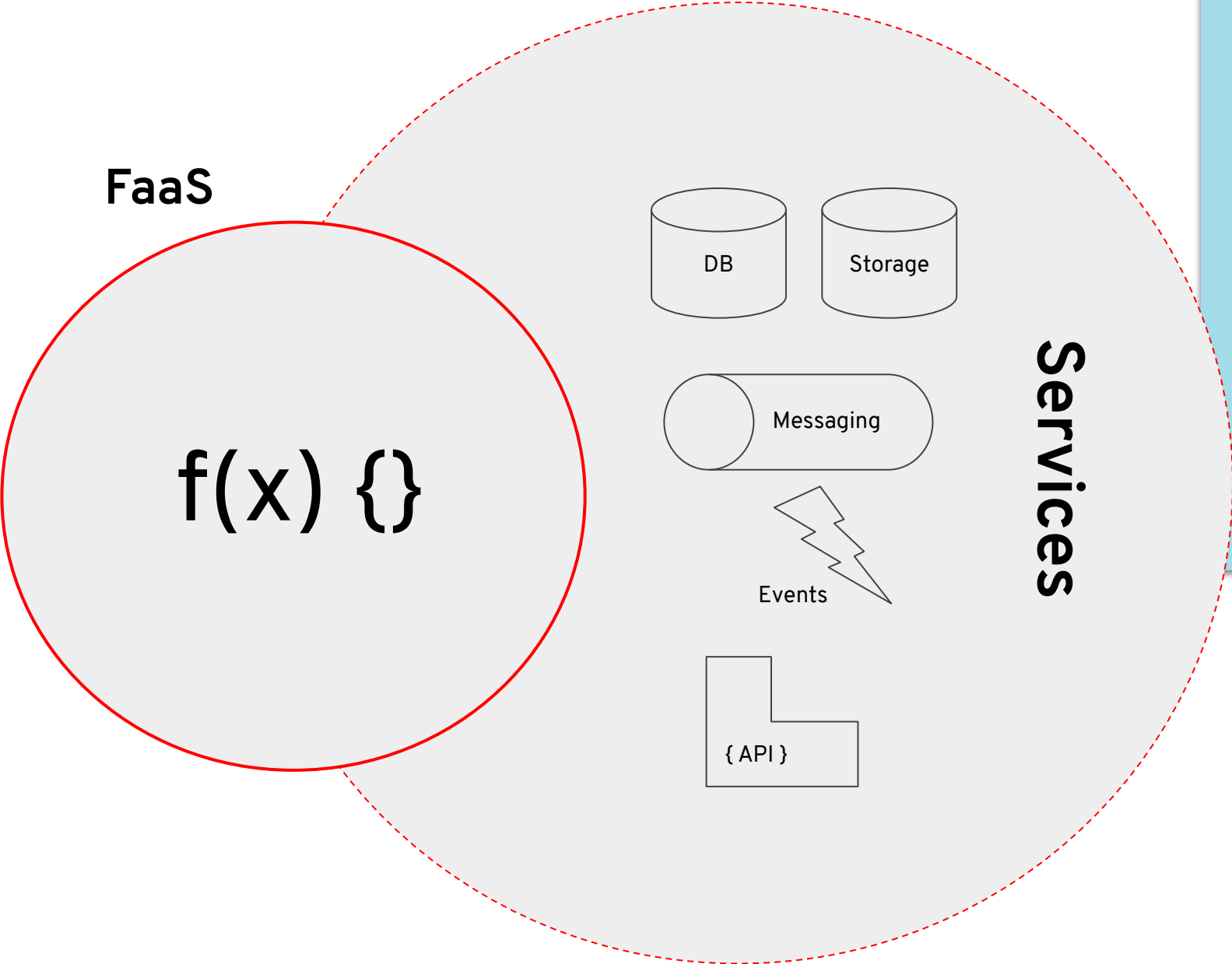


## Essentially, a service at rest

- Application code / function
- Deployed, to some infrastructure
- That requires no resources until needed
- Event driven
- And will scale up and down based on actual usage
- And then return to a state of rest when idle

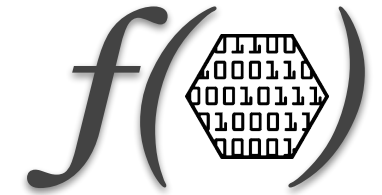


# Serverless



- User experience
- Services
- Debugging/IDE Integration
- API Gateway Integration
- Billing/Charging model
  - Per function call
  - Per execution time
  - Resource consumption

# Common use cases...



- ▶ Processing web hooks
- ▶ Scheduled tasks (a la cron)
- ▶ Data transformation
- ▶ Mobile image manipulation  
(compression, conversion, and so on)
- ▶ Voice packet to JSON transformation  
(Alexa, Cortana, and so on)
- ▶ Mobile video analysis (frame-grabbing)
- ▶ PDF generation
- ▶ Mobile/MBaaS /single-page apps
- ▶ Chat bots

**Web**

**Mobile**

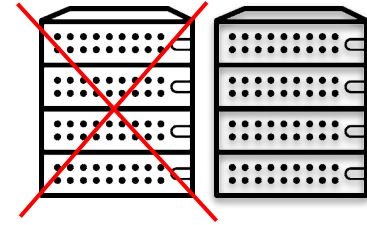
**IoT**

**DevOps Automation**

**Focus on convenience and  
business value, no distractions.**



# When not to use serverless



- *Real-time, ultra-low latency applications*
- *Long running tasks that can't be split into steps*
- *Advanced or complex observability and monitoring requirements*
- *Memory or CPU requirements are very demanding and specific*
- *Can't deal with cold-start...*

# Evolution of Serverless

# Evolution of Serverless

## 1.0

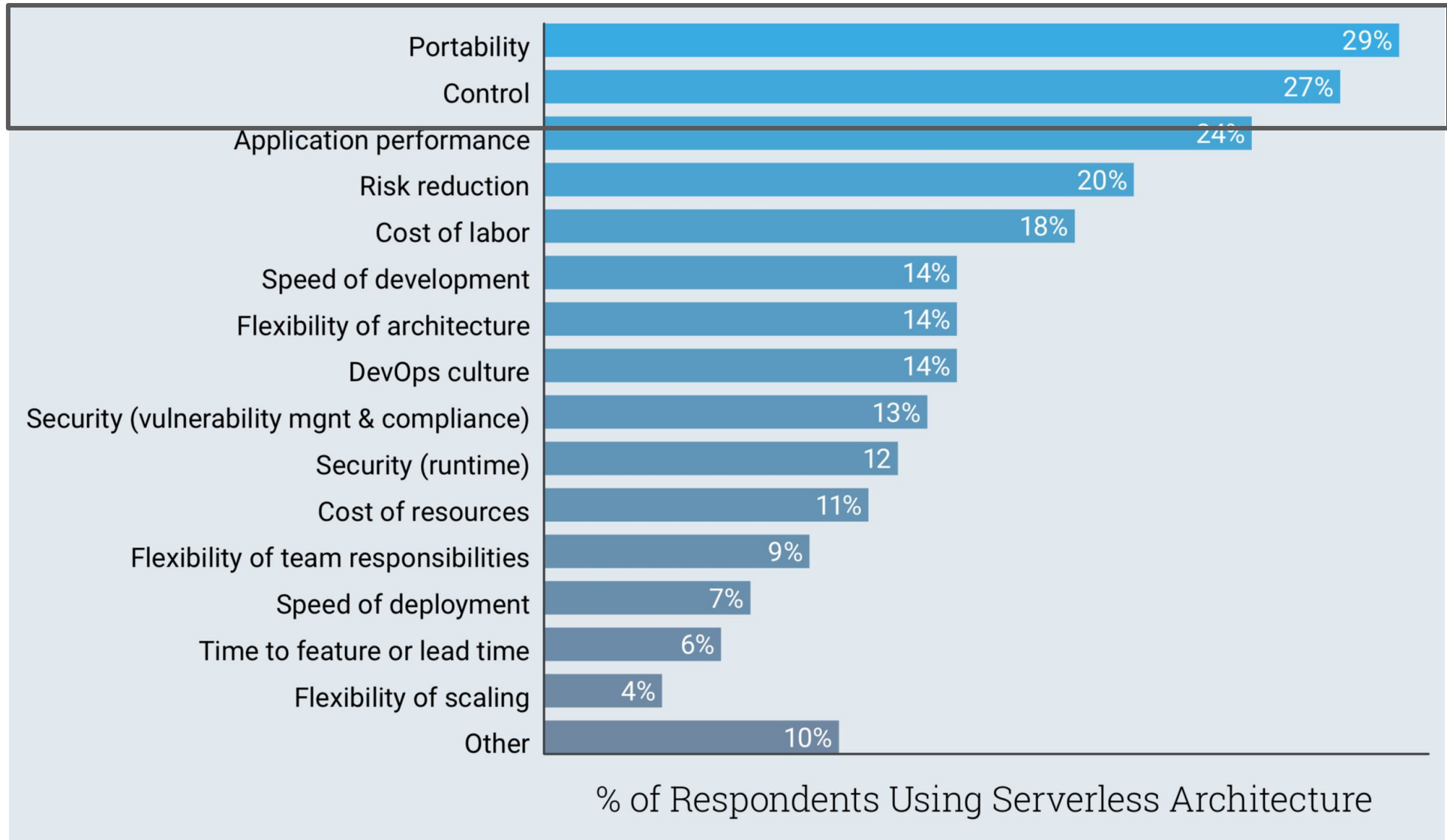
### AWS Lambda, Functions...

*Serverless 1.0 was built around the FaaS component and by other services such as API Gateways. The genesis of the current is general is available but far from ideal for general computing, and with potential candidates for improvements.*

- HTTP and other few Sources
- **Functions only**
- **Limited execution time (5 min)**
- No orchestration
- Limited local development experience



# Serverless Pain Points



# Evolution of Serverless

## 1.0

### AWS Lambda, Functions...

*Serverless 1.0 was built around the FaaS component and by other services such as API Gateways. The genesis of the current is general is available but far from ideal for general computing, and with potential candidates for improvements.*

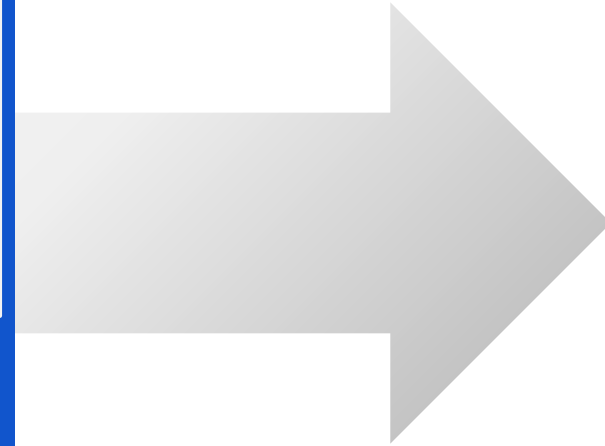
- HTTP and other few Sources
- **Functions only**
- **Limited execution time (5 min)**
- No orchestration
- Limited local development experience

## 1.5

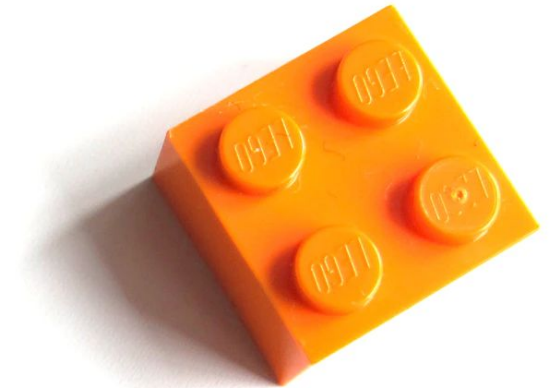
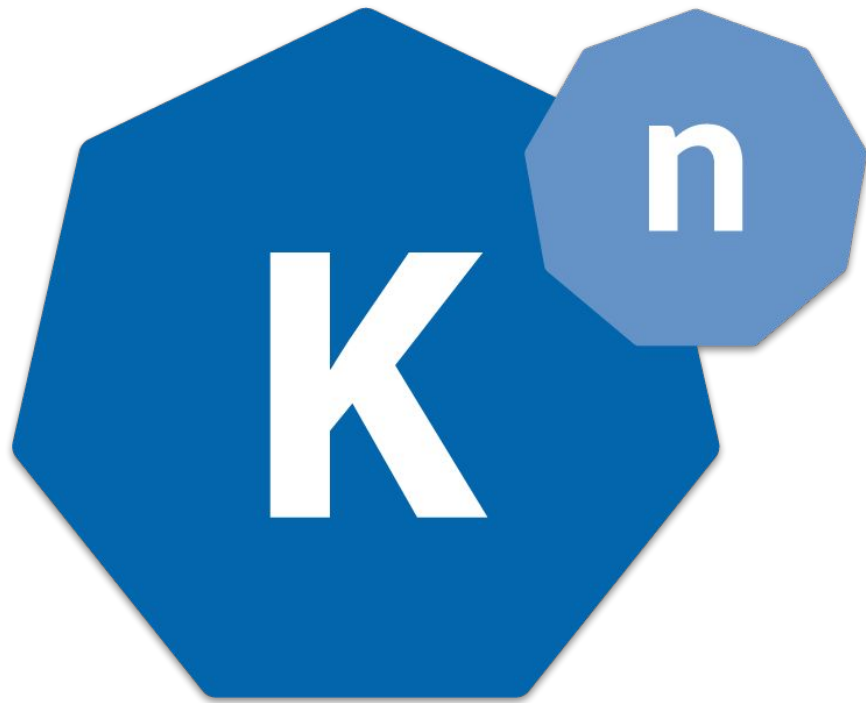
### Serverless Containers

*With the advent of Kubernetes, many frameworks and solutions started to auto-scale containers. Cloud providers created offerings using managed services completely abstracting Kubernetes APIs.*

- Red Hat joins Knative
- Kubernetes based auto-scaling
- **Microservices and Functions**
- Easy to debug & test locally
- **Polyglot & Portable**







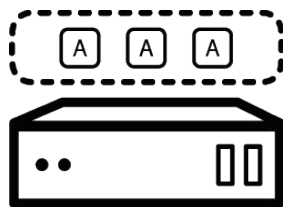


## SERVING

## EVENTING

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

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





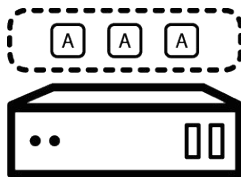
## (Openshift) Pipelines

Provides Kubernetes native modern resources for declaring CI/CD pipelines.



## Serving

An event-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.



# Evolution of Serverless

## 1.0

### AWS Lambda, Functions...

*Serverless 1.0 was built around the FaaS component and by other services such as API Gateways. The genesis of the current is general is available but far from ideal for general computing, and with potential candidates for improvements.*

- HTTP and other few Sources
- **Functions only**
- **Limited execution time (5 min)**
- No orchestration
- Limited local development experience

## 1.5

### Serverless Containers

*With the advent of Kubernetes, many frameworks and solutions started to auto-scale containers. Cloud providers created offerings using managed services completely abstracting Kubernetes APIs.*

- Red Hat joins Knative
- Kubernetes based auto-scaling
- **Microservices and Functions**
- Easy to debug & test locally
- **Polyglot & Portable**

## 2.0

### Integration & State

*The maturity and benefits of Serverless are recognized industry wide and providers start adding the missing parts to make Serverless suitable for general purpose workloads and used on the enterprise.*

- Basic state handling
- **Enterprise Integration Patterns**
- Advanced Messaging Capabilities
- **Blended with your PaaS**
- Enterprise-ready event sources

# Kubernetes

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
  labels:
    app: guestbook
spec:
  selector:
    matchLabels:
      app: guestbook
      tier: frontend
  replicas: 1
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
        - image: markusthoemmes/guestbook
          name: guestbook
          resources:
            requests:
              cpu: 100m
              memory: 100Mi
          env:
            - name: GET_HOSTS_FROM
              value: dns
          ports:
            - containerPort: 80
```

**~70 lines**

```
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: v1
kind: Service
metadata:
  name: frontend-service
  labels:
    app: guestbook
    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
```



# Knative

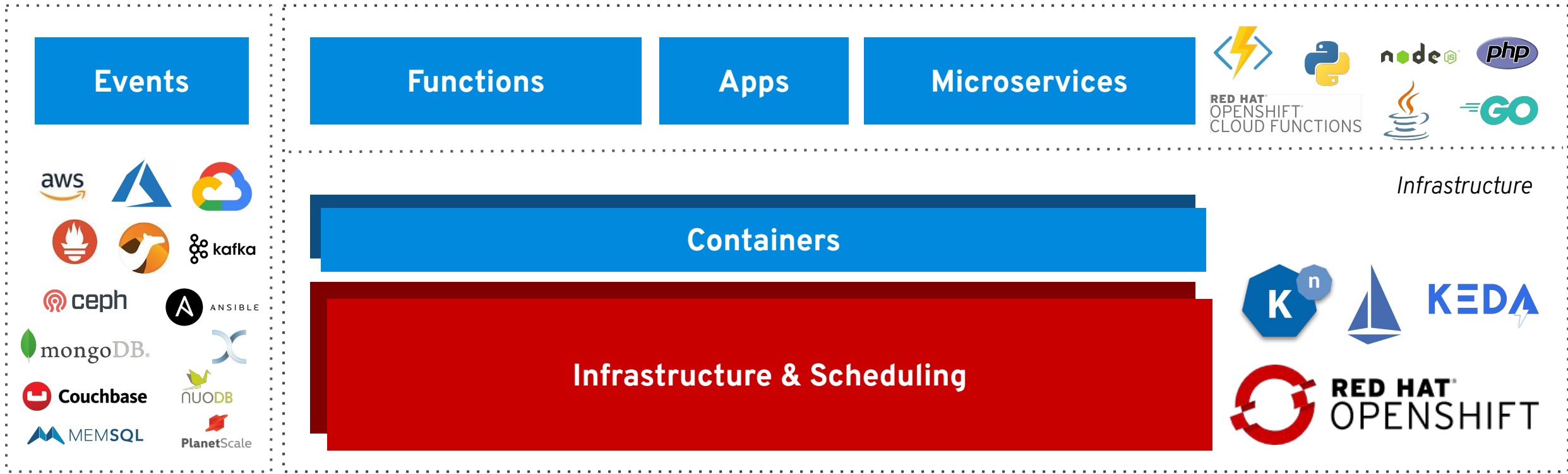
```
apiVersion: serving.knative.dev/v1alpha1
kind: Service
metadata:
  name: frontend
spec:
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
        - image: markusthoemmes/guestbook
          resources:
            requests:
              cpu: 100m
              memory: 100Mi
          env:
            - name: GET_HOSTS_FROM
              value: dns
          ports:
            - containerPort: 80
```

**22 lines**





# Microservices, Functions and Apps + Events = OpenShift Serverless

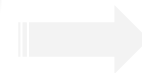


OperatorHub.io



# DEMO





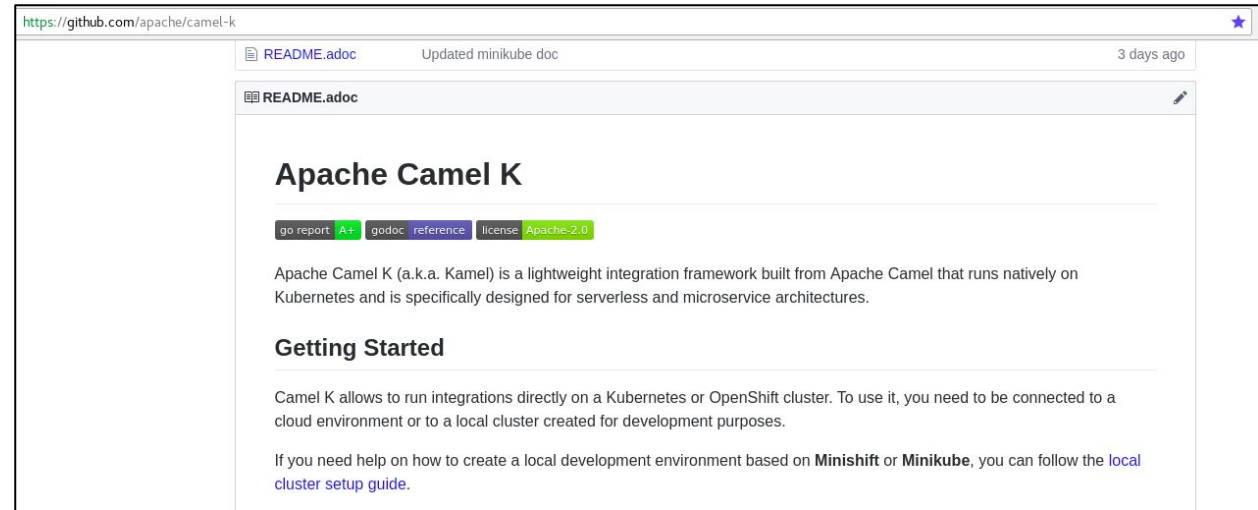
# QUARKUS

## Supersonic Subatomic Java

A Kubernetes Native Java stack tailored for OpenJDK HotSpot and GraalVM,  
crafted from the best of breed Java libraries and standards.

```
mvn io.quarkus:quarkus-maven-plugin:1.0.0.CR1:create \  
-DprojectId=org.acme \  
-DprojectId=getting-started \  
-DclassName="org.acme.quickstart.GreetingResource" \  
-Dpath="/hello"  
cd getting-started  
mvn package -Pnative -Dnative-image.docker-build=true  
kn service create gettingstarted-quarkus --image=markito/getting-started:v1
```

# APACHE CAMEL K



- ▶ **A platform for directly running integrations on Openshift and Kubernetes**
- ▶ Based on Operator SDK
- ▶ Apache-based, community-driven project
- ▶ A subproject of Apache Camel started on **August 31st, 2018**

<https://github.com/apache/camel-k>

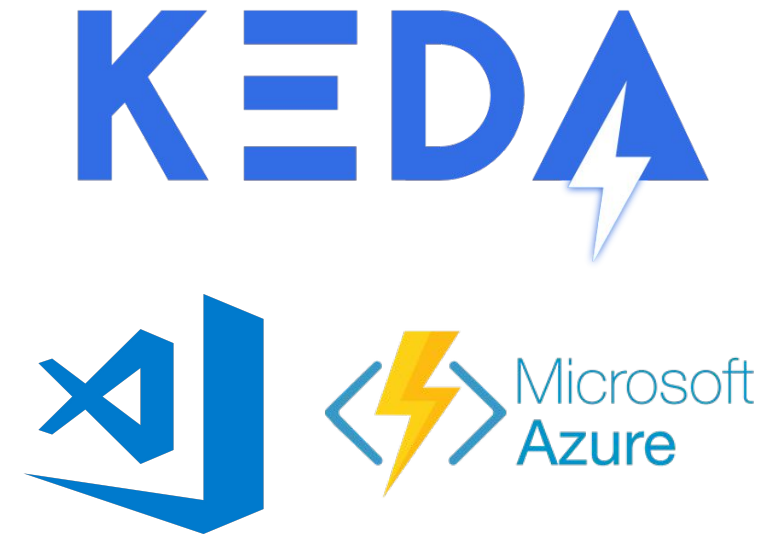
# Azure Functions & KEDA

## Key features

- Enable FaaS in OpenShift
- Familiar developer experience using VS Code and Azure CLI
- Polling based auto-scaling for Azure Queues, Kafka...
- Reuse Knative event sources, HTTP auto-scaling
- On premise or Any cloud.
- Familiar to Kubernetes users.

## Learn more

- <https://github.com/kedacore/keda>





# Kogito



*A continuation of Drools, jBPM and Optaplanner but completely redesigned to be cloud-native!*

## Next-gen Cloud-Native Business Automation

Cloud-Native Business Automation for building intelligent applications, backed by battle-tested capabilities



kubernetes



OPENSIFT



QUARKUS



OPERATOR  
FRAMEWORK

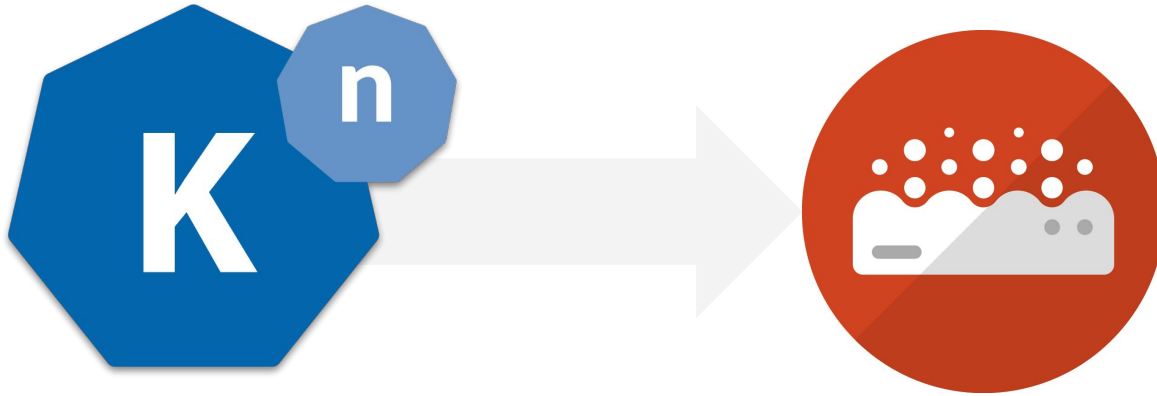


APACHE  
kafka



Grafana

COMMUNITY  
ONLY!



## Learn more

### OpenShift Serverless

Build and deploy serverless applications using an event-driven infrastructure on Red Hat® OpenShift®

### Tutorial

Get started with your serverless journey

### Knative Blog series

Knative: Serving your Serverless Services

<https://www.openshift.com/learn/topics/serverless>

The screenshot shows the Red Hat OpenShift Container Platform console. The main view displays a deployment overview for 'spring-petclinic-bchpw-deployment'. A large circular progress indicator shows '4 scaling to 10'. The deployment details include:

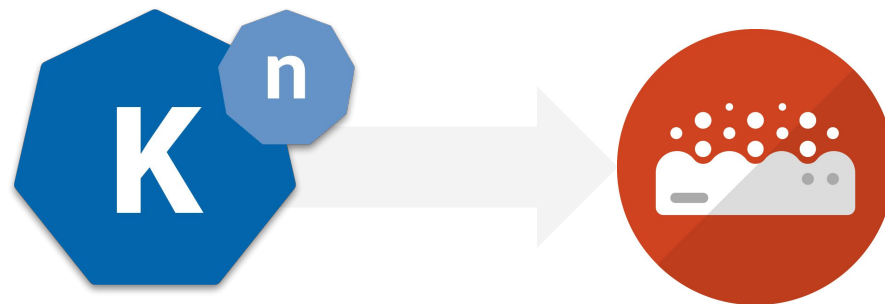
Name	Update Strategy
spring-petclinic-bchpw-deployment	RollingUpdate

Namespace	Max Unavailable
markito-rhte	25% of 10 pods

Labels	Max Surge
app=spring-petclinic-bchpw	25% greater than 10 pods
app.kubernetes.io/...=springBoo...	
app.kubernetes.io/...=spring-pe...	
-serving.knative....=spring-petcli...	
-serving.knative.dev/configurati...	
-serving.knat....=b8c3da91-d4cb-1...	


Other deployment details shown include 'Progress Deadline: 2m 0s' and 'Min Ready Seconds: Not Configured'. The console also shows a sidebar with navigation options like Developer, Topology, Builds, Pipelines, and Projects.

# Thank you



 [linkedin.com/company/red-hat](https://www.linkedin.com/company/red-hat)

 [facebook.com/redhatinc](https://www.facebook.com/redhatinc)

 [youtube.com/user/RedHatVideos](https://www.youtube.com/user/RedHatVideos)

 [twitter.com/RedHat](https://twitter.com/RedHat)