Cloud Agnostic Design Patterns and Tips
(for Serverless and Java)

Kevin Dubois,
Principal Developer Advocate, Red Hat
The Cloud

@kevindubois
Kevin Dubois

★ Principal Developer Advocate at Red Hat 🧙riters
★ Java Champion 🐾 Java Champion
★ Based in Belgium 🇧🇪
★ Speak English, Dutch, French, Italian 🗣️
★ Open Source Contributor (Quarkus, Camel, Knative, ..)
★ Community Member (BeJUG, BeCNCF)

@kevindubois@mastodon.social
youtube.com/@thekevindubois
linkedin.com/in/kevindubois
github.com/kdubois
@kevindubois.com
Cloud Computing:

Respond more quickly to demand

No provisioning/managing of hardware

High availability, Disaster Recovery, Resilience

Grow your application in a manageable way

Use only the resources you need!
(Some) Components of an Application Platform

- Developer Tools
- CI/CD
- Networking
- Monitoring
- Logging
- Container Registry
<table>
<thead>
<tr>
<th>Category</th>
<th>Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute</td>
<td>EC2, Lightsail, Lambda, Batch, Elastic Beanstalk, Serverless Application Repository, AWS Outposts, EC2 Image Builder, AWS App Runner, AWS SimSpace Weaver</td>
</tr>
<tr>
<td>Storage</td>
<td>S3, EFS, S3 Glacier, Storage Gateway, AWS Backup, AWS Elastic Disaster Recovery</td>
</tr>
<tr>
<td>Database</td>
<td>RDS, ElastiCache, Neptune, Amazon QLDB, Amazon DocumentDB, Amazon Keyspaces, Amazon TimeStream, DynamoDB, Amazon MemoryDB for Redis</td>
</tr>
<tr>
<td>Migration &amp; Transfer</td>
<td></td>
</tr>
<tr>
<td>Machine Learning</td>
<td>Amazon SageMaker, Amazon Augmented AI, Amazon CodeGuru, Amazon DevOps Guru, Amazon Comprehend, Amazon Forecast, Amazon Fraud Detector, Amazon Kendra, Amazon Personalize, Amazon Polly, Amazon Rekognition, Amazon Textract, Amazon Transcribe, Amazon Translate, AWS DeepComposer, AWS DeepRacer, AWS Panorama, Amazon Monitron, AWS HealthLake, Amazon Lookout for Vision, Amazon Lookout for Equipment, Amazon Lookout for Metrics, Amazon Lex, Amazon Comprehend Medical, AWS HealthOnics, Amazon Bedrock, AWS Health Imaging, Amazon Q</td>
</tr>
<tr>
<td>Analytics</td>
<td>Athena, Amazon Redshift, CloudSearch, Amazon OpenSearch Service, Kinesis, QuickSight, Data Pipeline, AWS Data Exchange, AWS Lake Formation</td>
</tr>
<tr>
<td>Front-end Web &amp; Mobile</td>
<td>AWS Amplify, AWS AppSync, Device Farms, Amazon Location Service</td>
</tr>
<tr>
<td>Application Integration</td>
<td>Step Functions, Amazon AppFlow, Amazon MQ, Simple Notification Service, Simple Queue Service, SWF, Managed Apache Airflow, Amazon EventBridge, AWS R2 B2 Data Interchange</td>
</tr>
<tr>
<td>Business Applications</td>
<td>Amazon Connect, Amazon Honeycode, Amazon Chime, Amazon Simple Email Service, Amazon WorkDocs, Amazon WorkMail, AWS Supply Chain, AWS AppFabric, AWS WAA, Amazon Chime SDK, Amazon One Enterprise, Amazon Pinpoint</td>
</tr>
<tr>
<td>End User Computing</td>
<td>WorkSpaces, AppStream 2.0</td>
</tr>
</tbody>
</table>
### Compute
Run scalable virtual machines and containers

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute Engine</td>
<td>VMs, GPUs, TPUs, disks</td>
</tr>
<tr>
<td>Kubernetes Engine</td>
<td>Managed Kubernetes / containers</td>
</tr>
<tr>
<td>VMware Engine</td>
<td>VMware as a service</td>
</tr>
<tr>
<td>Anthos</td>
<td>Enterprise hybrid multi-cloud platform</td>
</tr>
<tr>
<td>Batch</td>
<td>Jobs as a service</td>
</tr>
</tbody>
</table>

### Storage
Store long-term, short-term, VM, and Filestore securely

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloud Storage</td>
<td>Enterprise-ready object storage</td>
</tr>
<tr>
<td>Filestore</td>
<td>Fully managed NFS server</td>
</tr>
<tr>
<td>Storage Transfer</td>
<td>Secure and flexible way to move data</td>
</tr>
<tr>
<td>PoweredScale</td>
<td>Cloud-native enterprise-grade file service</td>
</tr>
<tr>
<td>NetApp Volumes</td>
<td>Fully Managed File Storage</td>
</tr>
</tbody>
</table>

### Analytics
Collect, store, process, and analyze large amounts of data

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BigQuery</td>
<td>Data warehouse/analytics</td>
</tr>
</tbody>
</table>
But...
What if ...

- Regulatory changes
- Outages
- Price changes, contract renegotiations
- Other vendor offers better hw/services
- New CIO/CTO
- Shadow IT
- ...

@kevindubois
So ...?
Hybrid / Multi Cloud!?
Building an application platform on each cloud

The required Parts

The AWS Car

The Azure Car

The Google Cloud Car

Monitoring | Service Mesh | CI/CD
---|---|---
CloudWatch | AppMesh | CodeBuild
ECR | CloudWatch | CodeStream
VPC, ELB, ALB
EKS
Amazon Linux 2

Monitoring | Open Service Mesh | Azure DevOps
---|---|---
ACR | Azure Monitor | Azure DevOps
VNET, ALB, Frontdoor, App gateway
AKS
Ubuntu

Monitoring | Anthos Service Mesh | CodeBuild
---|---|---
ACR | CloudWatch | CodeStream
VPC, CLB
GKE
Ubuntu / Container OS

3 different cars
- Different component versions
- Different life cycles
- Different support models
- Different developer and ops tooling

3 different drivers and pit crews needed
[Screams Internally]
Open Source & Cloud Native Ecosystem FTW!

Multicloud communication for Kubernetes

Skupper is a layer 7 service interconnect. It enables secure communication across Kubernetes clusters with no VPNs or special firewall rules.

With Skupper, your application can span multiple cloud providers, data centers, and regions.
What about Serverless?
Serverless

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

### What factors are leading you to overspend?

Select all that apply.

<table>
<thead>
<tr>
<th>Factor</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overprovisioning — for example, workloads using more resources than necessary</td>
<td>70%</td>
</tr>
<tr>
<td>Lack of individual or team-level awareness or responsibility</td>
<td>45%</td>
</tr>
<tr>
<td>Sprawl — such as resources not deactivated after use</td>
<td>43%</td>
</tr>
<tr>
<td>Technical debt — existing workloads not re-architected for scalability of cloud</td>
<td>43%</td>
</tr>
<tr>
<td>Lack of visibility and insight into consumption, budget and spending</td>
<td>40%</td>
</tr>
<tr>
<td>Presence of resource-hungry workloads</td>
<td>25%</td>
</tr>
<tr>
<td>Fluctuating consumption demands</td>
<td>23%</td>
</tr>
<tr>
<td>Poor planning and prediction on cloud consumption</td>
<td>23%</td>
</tr>
<tr>
<td>Absence of centralized, consistent or standardized processes and/or tools for insight and action across all our cloud providers</td>
<td>20%</td>
</tr>
<tr>
<td>Availability of a self-service infrastructure</td>
<td>15%</td>
</tr>
</tbody>
</table>

Traditional Deployments

overprovisioned & wasting resources

underprovisioned & unhappy users!
Serverless

Use only what you need!
Java & Serverless
JVM Scaling

Not meeting SLA
QUARKUS

Supersonic. Subatomic. Java

@kevindubois
The Quarkus Way
The Quarkus Way enables Native Compilation OOTB

Build Time

Package

Native

JVM

@kevindubois
Supersonic, Subatomic Java

- **Quarkus + Native (via GraalVM)**: 0.016 Seconds
- **Quarkus + JVM (via OpenJDK)**: 0.943 Seconds
- **Traditional Cloud-Native Stack**: 4.3 Seconds

- **Quarkus + Native (via GraalVM)**: 12 MB
- **Quarkus + JVM (via OpenJDK)**: 73 MB
- **Traditional Cloud-Native Stack**: 136 MB
Java warmup time

Application Runtime Performance

- Framework A
- Quarkus JVM
- Quarkus Native
- SLA Throughput

Throughput (req/sec)

Time
AWS Lambda, Functions...

Built around the FaaS components and other services such as API Gateways. It enabled a variety of use cases but it is far from ideal for general computing and with room for improvements.

➔ HTTP and other few Sources
➔ Functions only
➔ Limited execution time (5 min)
➔ No orchestration
➔ Limited local development experience
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import java.util.Map;

// Handler value: example.Handler
public class Handler implements RequestHandler<String, Void> {

    @Override
    public Void handleRequest(String event, Context context) {
        LambdaLogger logger = context.getLogger();
        logger.log("EVENT TYPE: " + event.getClass());
        return null;
    }
}
Writing cloud agnostic functions with Java & Quarkus
quarkus create app funqy \\ -x quarkus-funqy-amazon-lambda
API GATEWAY IS RESPONSIBLE FOR OVER HALF OF ALL LAMBDA FUNCTION INVOCATIONS

Source: Datadog
### 1.0

**AWS Lambda, Functions...**

Built around the FaaS components and other services such as API Gateways. It enabled a variety of use cases but it is far from ideal for general computing and with room 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 containers & Kubernetes, many frameworks and solutions started to auto-scale containers. Cloud providers created offerings using managed services completely abstracting Kubernetes APIs.

- Fargate, Cloud Run, Container Instances
- Knative, KEDA, etc
- Kubernetes based auto-scaling
- Microservices and Functions
- Easier to debug & test locally
- Polyglot & Portable

---

**Serverless is evolving...**
Knative

https://github.com/knative
**Knative**

Knative is an **Open Source, Cloud Agnostic** Solution to build Serverless and Event Driven Applications on Kubernetes

<table>
<thead>
<tr>
<th>Containers made easy</th>
<th>Ready for the Hybrid Cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simplified developer experience to deploy applications/code on serverless containers <strong>abstracting infrastructure</strong> &amp; focusing on what matters.</td>
<td>Truly portable serverless running anywhere Kubernetes runs, that is on-premises or on any public cloud. Leverage data locality and SaaS when needed.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Immutable revisions</th>
<th>Any programming language</th>
<th>Event Driven Architectures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deploy new features: performing canary, A/B or blue-green testing with gradual traffic rollout with no sweat and following best practices.</td>
<td>Use any programming language or runtime of choice. From Java, Python, Go and JavaScript to Quarkus, SpringBoot or Node.js.</td>
<td>Build <strong>loosely coupled &amp; distributed apps</strong> connecting with a variety of built-in or third-party event sources or connectors powered by Operators.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Automatic scaling</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>No need to configure number of replicas, or idling. <strong>Scale to zero</strong> when not in use, auto scale to thousands during peak, with built-in reliability and fault-tolerance.</td>
<td></td>
</tr>
</tbody>
</table>
$ kn func create -l quarkus myfunc
$ kn func deploy

$ kn service create myservice --image=xyz

github.com/serverless-java-in-action/examples
AWS Lambda, Functions...

Built around the FaaS components and other services such as API Gateways. It enabled a variety of use cases but it is far from ideal for general computing and with room for improvements.

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

Serverless Containers

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

- Fargate, Cloud Run, Container Instances
- Knative, KEDA, etc
- Kubernetes based auto-scaling
- Microservices and Functions
- Easier to debug & test locally
- Polyglot & Portable

Integration & State

The maturity and benefits of Serverless are recognized industry wide and it adds the missing parts to make pattern 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
- Solutions and outcome focused
Knative has two main components that empower teams working with Kubernetes. Serving and Eventing work together to automate and manage tasks and applications.
Knative Eventing

Eventing is a set of APIs for routing events from **Producers** to **Consumers** (known as **Sinks**)

**CloudEvent** specification allows for the creation of Serverless components that are driven by Event rather than Traffic
CloudEvents

CNCF graduated project - https://cloudevents.io/

Provides a common event schema => Interoperability, portability

Extensible through extension attributes

SDKs for different programming languages

Protocol-agnostic (HTTP, AMQP, MQTT, ...)

Wide adoption
Usage Patterns
Usage Patterns

Channel and Subscription

Knative Event Source

Sink

Channel

Subscription

Sink

Knative Service A (ksvc)

Subscription

Kubernetes Service B (svc)

Cloud Event Message
Usage Patterns

Broker and Trigger

Knative Event Source

Sink

Knative Eventing Trigger

Knative Service Green (ksvc)

Kubernetes Service Red (svc)

Subscription

Filtered messages

@kevindubois
## Who wins?

<table>
<thead>
<tr>
<th>Position</th>
<th>Name</th>
<th>Count</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IntelliJ</td>
<td>1</td>
<td>Vote for IntelliJ!</td>
</tr>
<tr>
<td>2</td>
<td>VScode</td>
<td>1</td>
<td>Vote for VScode!</td>
</tr>
<tr>
<td>3</td>
<td>Eclipse</td>
<td>0</td>
<td>Vote for Eclipse!</td>
</tr>
<tr>
<td>4</td>
<td>Vim</td>
<td>0</td>
<td>Vote for Vim!</td>
</tr>
<tr>
<td>5</td>
<td>Other</td>
<td>0</td>
<td>Vote for Other!</td>
</tr>
<tr>
<td>6</td>
<td>OpenShift Dev Spaces</td>
<td>0</td>
<td>Vote for OpenShift Dev Spaces!</td>
</tr>
</tbody>
</table>

![Graph showing the comparison between IntelliJ, VScode, Eclipse, Vim, Other, and OpenShift Dev Spaces]
Serverless + AI?

https://knative.dev/docs/about/case-studies/deepc/
Wrapping it up...

- Cloud providers offer a LOT of cool stuff
- We need to be mindful of cloud lock-in
- Serverless is much more than just FAAS
- Use Open Source when you can, proprietary services when you must
- If you find yourself limited by Open Source solutions, contribute and participate!
Free Developer e-Books & tutorials!
developers.redhat.com/eventtutorials
OpenShift Sandbox:

We changed the default deployment type to Serverless
github.com/serverless-java-in-action
Serverless Java in Action
Kevin Dubois & Daniel Oh
Thank you!

@kevindubois@mastodon.social

@kevindubois.com

youtube.com/@thekevindubois

linkedin.com/in/kevindubois

github.com/kdubois