Lessons Learned: Building Scalable & Elastic Akka Clusters on Google Managed Kubernetes

- Timo Mechler & Charles Adetiloye
About MavenCode

MavenCode is a Data Analytics software company offering training, product development, and consulting services in the following areas:

- Provisioning Scalable Data Processing Pipelines and Cloud Infrastructure Deployment
- Development & Deployment of Machine Learning and Artificial Intelligence Platforms
- Streaming and Big Data Analytics - IoT and Sensors
About The Presenters

**Timo Mechler (Architect & Product Manager)**
Decade of experience in the energy commodity markets with particular focus building out scalable research platforms for commodities trading (data collection, data analysis, data modeling).

**Charles Adetiloye (Lead Data Engineer)**
Over a decade worth of experience consulting and implementing large scale distributed data processing software platforms across different industry verticals. Previously worked/consulted with Lightbend, Twitter, Monsanto, Starbucks, and a few other startups and Fortune 500 companies.
Moving From “Proactive” to “Reactive”!

- Beefed Up Servers
- Difficult to Scale
- Slow Network IO
- Few Concurrent Processes
- Deployment Nightmare

- Virtualized Commodity Hardware
- More Distributed Spread Out Nodes
- Improved Network IO
- Network Admin Functional DevOps Team

https://www.reactivemanifesto.org/
Containerization & Cloud Orchestration

Application Stack
- Scala + Akka, Some Go & Python
- Alpine Image Dockerized
- Akka, Clustering, Remoting, HTTP, Alpakka

<table>
<thead>
<tr>
<th>Usability</th>
<th>Docker Swarm</th>
<th>Mesos</th>
<th>Kubernetes</th>
</tr>
</thead>
<tbody>
<tr>
<td>🧐</td>
<td>🧐</td>
<td>🧐</td>
<td>🧐</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stability</th>
<th>Docker Swarm</th>
<th>Mesos</th>
<th>Kubernetes</th>
</tr>
</thead>
<tbody>
<tr>
<td>🧐</td>
<td>🧐</td>
<td>😊</td>
<td>🧐</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Feature Sets</th>
<th>Docker Swarm</th>
<th>Mesos</th>
<th>Kubernetes</th>
</tr>
</thead>
<tbody>
<tr>
<td>😞</td>
<td>🧐</td>
<td>😊</td>
<td>🧐</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Community/EcoSystem</th>
<th>Docker Swarm</th>
<th>Mesos</th>
<th>Kubernetes</th>
</tr>
</thead>
<tbody>
<tr>
<td>😞</td>
<td>🧐</td>
<td>😊</td>
<td>🧐</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Here To Stay?</th>
<th>Docker Swarm</th>
<th>Mesos</th>
<th>Kubernetes</th>
</tr>
</thead>
<tbody>
<tr>
<td>😞</td>
<td>🧐</td>
<td>😊</td>
<td>🧐</td>
</tr>
</tbody>
</table>

We Work With All 3 Cloud Services And They're All Great!!!
But We think Google Cloud Platform (GCP) stands out:
- Kubernetes was started at Google
- If you are doing AI & ML stuff, GCP integration is the best
- From a cost perspective with GCP you save a few $$$
Why Did We Go Reactive With Akka?

- High Performance, Resilience and Scalability
- Loosely Coupled Messaging System
- Active Open Source Developer Community
- Battle Tested Framework, Proven Use Cases, Matured but Still Improving (since 2009)
Scalable DataPipeline

1. Events are ingested - Satellite, Telemetry, IoT, etc.
2. Events Processing Queue, Google Pub-Sub/Kafka
3. Schema Registry for Event Validation
4. Near Real-time Continuously Streaming Events
5. Batch Rollup JOB - Time or Size Rotation: TimeStamped
6. DataStore -> Parquet Compressed on Google Storage or Amazon S3
7. ML Models Generated and Versioned -> TensorFlow, MXNet, Spark MLlib
8. Near Real-time Inferencing and Predictive Intelligence
How Do You Scale Your Akka Cluster Pipeline?

- Time-Based (GeoSpatial) Scheduled Scaling
  - Event `always` happen at certain times of the day
  - We have a rough idea of traffic seasonality, and we can project the future needs
  - Happens across Timezones, we can always skew our Cluster Workload (Time, Location)

- Surge-Based Scaling
  - Sudden spike in traffic, Due to some external factor or influencer
  - Delayed Delivery or Batched Delivery
Time-Based Scheduling with Akka Cluster + Kubernetes

1. Config a Cluster Aware Group Router
   - akka.actor.deployment
     - router = round-robin-group
     - routee.paths = ["/telematicsService/ComputeWorkerNode"]
     - cluster
       - enabled = on
       - allow-local-routees = off
       - use-roles = ["computeWorkRate"]

2. Role Out the StatefulSet with the right Akka Actor Role

BasketBall Rotation Strategy!!!
Surge/Spike-Based Scaling with Akka-Cluster & Kubernetes

Using Cluster-Aware Pool Routers

1. **Startup the Pool Router + Configure it to Startup on Member Nodes in the Cluster**

   ```scala
   akka.actor.deployment {
     router = round-robin-pool
     routee.paths = ["/telematicsService/singleton/SignUpNode"]
   }
   
   cluster {
     enabled = on
     allow-local-routees = off
     max-nr-instances-per-node = 3
     use-roles = ["AppRegistration"]
   }
   
   }```

2. **Startup a Pod with the right role in AkkaConfig, Configure it for Horizontal Scalability with K8s**

   ```yaml
   metrics:
   - minReplicas: 1
     maxReplicas: 10
     type: Resources
     resource: CPU
     target:
   ```

3. **During Spike in Traffic, Pods will be automatically scaled out with the right role config**

   HorizontalPod Scaling→
Cluster Bootstrap with Akka Management & Service Discovery

1. Central “Glue” point for all Akka Management extensions + Management endpoints
2. Management Endpoints show the status of the Cluster
3. Akka Service Discovery is like a “LEGO tool box”
Cluster Bootstrap + Service Discovery with Kubernetes API

1. AkkaManagement Service discovery needs to grab initial seed nodes `/bootstrap/seed-nodes`
2. In our case, Kubernetes is used for discovery by querying for all pods with matching `pod-labels` in the config.
3. The Node Probes for existing Cluster, if YES it will Join, if NO it will create a new cluster.
4. Some Process is Repeated on Other Nodes and if all succeed, then we have a cluster!

AkkaManagement (system).start

ClusterBootstrap (system).start

//discovery-config
akka.discovery.kubernetes-api {
  pod-label-selector="clusterName=%s"
  pod-namespace="demo_telematics"
  api-ca-path="/app/opt/telematics/serviceaccount/ca.crt"
  api-ca-token="/app/opt/telematics/serviceaccount/token"
  api-service-host-env-name="KUBERNETES_SERVICE_HOST"
  api-service-port-env-name="KUBERNETES_SERVICE_PORT"
}

//management-config
akka.management.cluster.bootstrap {
  contact-point-discovery{
    service-name="telematics"
    discovery-method=akka.discovery.kubernetes-api
  }
}

Looking good so far!
But How do I get started?
3-Step Deployment Process

1. SBT build/package/dockerize your AKKA code
2. SBT Publish to Docker Registry.
3. Helm Deploy to Minikube (Dev/Test) or GKE (PROD)
Deployments with Helm Charts

We Use HELM for Managing:

- Container Packing and Deployment on Kubernetes in Different Environments
- Upgrading and Versioning Container Deployments

![Diagram of Helm Chart setup]

- Users go to:
  - app1.rxdemo.com
  - app2.rxdemo.com

- e.g Google Cloud Layer 7 Load Balancer
  - Looks up routing rules to route to the correct services

- Kubernetes Service Deployments
- Kubernetes POD Deployments
Quick Demo - Telematics Event Processor on Google Cloud

**TELEMATIC EVENTS**
- Tire Pressure
- Location Info
- Fuel Consumption
- Average Speed

**WEATHER INFO**

**REACTIVE PIPELINE**
- ClusterSingletonManager
- ClusterSingletonProxy

**ML PIPELINE**
- Model Versions: A|B|C
- Predictive Analysis
- Google Storage
- BigQuery

**SCALABLE PUB-SUB MESSAGE QUEUE**
GoogleCloud Kubernetes Setup for Stateful Akka Deployment

1. Create Multi-Zone Cluster

```bash
gcloud container clusters create telematics-rx18-cluster --zone us-central1-a --node-locations us-central1a, us-central1b, us-central1c
```

2. Create NameSpace for Your Akka Clusters

```bash
kubectl create namespace ns-telematics
```

3. Create Service Account

```bash
kubectl create serviceaccount sa-telematics -n ns-telematics

kubectl get sa-telematics -o json -n ns-telematics | jq -r .secrets[].name
```

4. Grab Service Account Certificate & Token

```bash
kubectl get secret sa-telematics-token-4478c -o json -n ns-telematics | jq -r '.data["ca.crt"]' | base64 -d > ca.crt

kubectl get secret sa-telematics-token-4478c -o json -n ns-telematics | jq -r '.data["token"]' | base64 -d > token
```

5. Grant the Right Privilege for the `sa-telematics` Service Account to Query PODs in the namespace

```bash
kubectl -n kube-system create clusterrolebinding rolebind-telematics --clusterrole=cluster-admin --serviceaccount=ns-telematics:sa-telematics
```
Lessons Learned

- With the growing number of interconnected devices generating data, infrastructure that can handle elastic data loads is more important than ever

- Kubernetes is a stable and continually growing container orchestration framework with an active development support community

- Deployment of Akka on Kubernetes is straightforward and helps avoid pitfalls related to scalability latency, and reliance on an external system for orchestration

- If you're not heavily invested in other platforms yet and looking to build a scalable backend + AI & ML integration down the road, it's worth checking out Google Cloud
Q & A

Special Thank You’s To:
- Reactive Summit Organizers
- Akka Team & Contributors
- Google Cloud

Contact Information:
Web: www.mavencode.com
Email: info@mavencode.com
Tel: +1 (682) 268-0571
Twitter: @mavencodeapps