How Microservices and Serverless Computing Enable the Next Gen of Machine Intelligence

Jon Peck
Full-Spectrum Developer & Advocate

Algorithmia
Making state-of-the-art algorithms discoverable and accessible to everyone

jpeck@algorithmia.com
@peckjon
Algorithmia.com
The Problem: ML is in a huge growth phase, difficult/expensive for DevOps to keep up

**Initially:**
- A few models, a couple frameworks, 1-2 languages
- Dedicated hardware or VM Hosting
- IT Team for DevOps
- High time-to-deploy, manual discoverability
- Few end-users, heterogenous APIs (if any)

**Pretty soon...**
- > 5,000 algorithms (50k versions) on many runtimes / frameworks
- > 60k algorithm developers: heterogenous, largely unpredictable
- Each algorithm: 1 to 1,000 calls/second, a lot of variance
- Need auto-deploy, discoverability, low (15ms) latency
- Common API, composability, fine-grained security
The Need: an “Operating System for AI”

AI/ML scalable infrastructure on demand + marketplace

- Function-as-a-service for Machine & Deep Learning
- Discoverable, live inventory of AI via APIs
- Anyone can contribute & use
- Composable, Monetizable
- Every developer on earth can make their app intelligent
General-purpose computing had a long evolution, as we learned what the common problems were / what abstractions to build. AI is in the earlier stages of that evolution.

An Operating System:

• Provides common functionality needed by many programs
• Standardizes conventions to make systems easier to work with
• Presents a higher level abstraction of the underlying hardware
Use Case

Jian Yang made an app to recognize food “SeeFood”
Use Case

He deployed his trained model to a GPU-enabled server
Use Case

The app is a hit!
Use Case

... and now his server is overloaded.
Characteristics of AI

- Two distinct phases: training and inference
- Lots of processing power
- Heterogenous hardware (CPU, GPU, FPGA, TPU, etc.)
- Limited by compute rather than bandwidth
- “Tensorflow is open source, scaling it is not.”
TRAINING

OWNER: Data Scientists

- Long compute cycle
- Fixed load (Inelastic)
- Stateful
- Single user
TRAINING

OWNER: Data Scientists
- Long compute cycle
- Fixed load (Inelastic)
- Stateful
- Single user

Analogous to dev tool chain. Building and iterating over a model is similar to building an app.

Metal or VM
Analogous to dev tool chain. Building and iterating over a model is similar to building an app.

**TRAINING**

**OWNER: Data Scientists**
- Long compute cycle
- Fixed load (Inelastic)
- Stateful
- Single user

**INFERENCE**

**OWNER: DevOps**
- Short compute bursts
- Stateless
- Elastic
- Multiple users

Metal or VM
TRAINING

OWNER: Data Scientists

- Long compute cycle
- Fixed load (Inelastic)
- Stateful
- Single user

Analogous to dev tool chain. Building and iterating over a model is similar to building an app.

INFERENGE

OWNER: DevOps

- Short compute bursts
- Stateless
- Elastic
- Multiple users

Analogous to an OS. Running concurrent models requires task scheduling.

Metal or VM
## TRAINING

**OWNER: Data Scientists**
- Long compute cycle
- Fixed load (Inelastic)
- Stateful
- Single user

**Analogous to dev tool chain.** Building and iterating over a model is similar to building an app.

## INFERENCCE

**OWNER: DevOps**
- Short compute bursts
- Stateless
- Elastic
- Multiple users

**Analogous to an OS.** Running concurrent models requires task scheduling.

Metal or VM

Containers
### TRAINING

**OWNER:** Data Scientists

- Long compute cycle
- Fixed load (Inelastic)
- Stateful
- Single user

**Analogous to dev tool chain.**
Building and iterating over a model is similar to building an app.

Metal or VM

### INFEERENCE

**OWNER:** DevOps

- Short compute bursts
- Stateless
- Elastic
- Multiple users

**Analogous to an OS.**
Running concurrent models requires task scheduling.

Containers
Kubernetes
TRAINING

OWNER: Data Scientists

- Long compute cycle
- Fixed load (Inelastic)
- Stateful
- Single user

Analogous to dev tool chain.
Building and iterating over a model is similar to building an app.

Metal or VM

INFERENCEN

OWNER: DevOps

- Short compute bursts
- Stateless
- Elastic
- Multiple users

Analogous to an OS.
Running concurrent models requires task scheduling.
**MICROSERVICES**: the design of a system as independently deployable, loosely coupled services.

**ADVANTAGES**
- Maintainable, Scalable
- Software & Hardware Agnostic
- Rolling deployments

**SERVERLESS**: the encapsulation, starting, and stopping of singular functions per request, with a just-in-time-compute model.

**ADVANTAGES**
- Elasticity, Cost Efficiency
- Concurrency
- Improved Latency
Why Serverless - Cost Efficiency

Jian Yang’s “SeeFood” is most active during lunchtime.
# Traditional Architecture - Design for Maximum

40 machines 24 hours. $648 * 40 = $25,920 per month

<table>
<thead>
<tr>
<th>Time</th>
<th>Calls per Second</th>
</tr>
</thead>
<tbody>
<tr>
<td>12 AM</td>
<td>5</td>
</tr>
<tr>
<td>02 AM</td>
<td>10</td>
</tr>
<tr>
<td>04 AM</td>
<td>20</td>
</tr>
<tr>
<td>06 AM</td>
<td>30</td>
</tr>
<tr>
<td>08 AM</td>
<td>40</td>
</tr>
<tr>
<td>10 AM</td>
<td>50</td>
</tr>
<tr>
<td>12 PM</td>
<td>55</td>
</tr>
<tr>
<td>02 PM</td>
<td>60</td>
</tr>
<tr>
<td>04 PM</td>
<td>65</td>
</tr>
<tr>
<td>06 PM</td>
<td>70</td>
</tr>
<tr>
<td>08 PM</td>
<td>75</td>
</tr>
<tr>
<td>10 PM</td>
<td>80</td>
</tr>
</tbody>
</table>

**Max calls/s**

- 80 (10 PM)
- 75 (08 PM)
- 70 (06 PM)
- 65 (04 PM)
- 60 (02 PM)
- 55 (12 PM)
- 50 (10 AM)
- 40 (08 AM)
- 30 (06 AM)
- 20 (04 AM)
- 10 (02 AM)
- 5 (12 AM)

**Avg calls/s**

- 5 (12 AM)
- 10 (02 AM)
- 20 (04 AM)
- 30 (06 AM)
- 40 (08 AM)
- 50 (10 AM)
- 60 (12 PM)
- 65 (02 PM)
- 70 (04 PM)
- 75 (06 PM)
- 80 (08 PM)
- 55 (10 PM)

**GPU Server Instances**

- 160 (10 PM)
- 140 (08 PM)
- 120 (06 PM)
- 100 (04 PM)
- 80 (02 PM)
- 60 (12 PM)
- 40 (04 AM)
- 20 (02 AM)

40 machines 24 hours. $648 * 40 = $25,920 per month
Autoscale Architecture - Design for Local Maximum

19 machines 24 hours. $648 * 40 = $12,312 per month

Max calls/s

Avg calls/s

12 AM 02 AM 04 AM 06 AM 08 AM 10 AM 12 PM 02 PM 04 PM 06 PM 08 PM 10 PM

Calls per Second

GPU Server Instances
Serverless Architecture - Design for Minimum

Avg. of 21 calls / sec, or equivalent of 6 machines. $648 * 6 = $3,888 per month
Why Serverless - Concurrency
Why Serverless - Improved Latency

Portability = Low Latency
Almost there! We also need:
GPU Memory Management, Job Scheduling, Cloud Abstraction,
Discoverability, Authentication, Logging, etc.
Elastic Scale

Cloud Region #1

Worker xN

Docker (algorithm #1)
...
Docker (algorithm #n)

Cloud Region #2

Worker xN

Docker (algorithm #1)
...
Docker (algorithm #n)
Elastic Scaling with Intelligent Orchestration

Knowing that:
- Algorithm A always calls Algorithm B
- Algorithm A consumes X CPU, X Memory, etc
- Algorithm B consumes X CPU, X Memory, etc

Therefore we can slot them in a way that:
- Reduce network latency
- Increase cluster utilization
- Build dependency graphs
Composability

Composability is critical for AI workflows because of data processing pipelines and ensembles.

cat file.csv | grep foo | wc -l
Cloud Abstraction - Storage

# No storage abstraction
s3 = boto3.client("s3")
obj = s3.get_object(Bucket="bucket-name", Key="records.csv")
data = obj["Body"].read()

# With storage abstraction
data = client.file("blob://records.csv").get()

s3://foo/bar
blob://foo/bar
hdfs://foo/bar
dropbox://foo/bar
etc.
## Cloud Abstraction

<table>
<thead>
<tr>
<th></th>
<th>Amazon Web Services</th>
<th>Google Cloud Platform</th>
<th>Microsoft Azure</th>
<th>OpenStack</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Compute</strong></td>
<td>EC2</td>
<td>CE</td>
<td>VM</td>
<td>Nova</td>
</tr>
<tr>
<td><strong>Autoscaling</strong></td>
<td>Autoscaling Group</td>
<td>Autoscaler</td>
<td>Scale Set</td>
<td>Heat Scaling Policy</td>
</tr>
<tr>
<td><strong>Load Balancing</strong></td>
<td>Elastic Load Balancer</td>
<td>Load Balancer</td>
<td>Load Balancer</td>
<td>LBaaS</td>
</tr>
<tr>
<td><strong>Remote Storage</strong></td>
<td>Elastic Block Store</td>
<td>Persistent Disk</td>
<td>File Storage</td>
<td>Block Storage</td>
</tr>
</tbody>
</table>

Partial Source: Sam Ghods, KubeConf 2016
An Operating System for AI: the “AI Layer”

- **Discoverability, Authentication, Instrumentation, etc.**
- **Elastic Scale**
  - Prioritize and automatically optimize execution of concurrent short-lived jobs.
- **Runtime Abstraction**
  - Support any programming language or framework, including interoperability between mixed stacks.
- **Cloud Abstraction**
  - Provide portability to algorithms, including public clouds or private clouds.

Shell & Services

Kernel
Discoverability: an App Store for AI
Algorithmia’s OS for AI: *discover* a model

1. **Discover a model**

   - AppStore-like interface
   - Categorized, tagged, rated
   - Well-described
     (purpose, source, API)
Algorithmia’s OS for AI: \textit{execute} a model

2. Execute from any language

- Raw JSON, or lang stubs
- Common syntax
- Autoscaled elastic cloud-exec
- Secure, isolated
- Concurrent, orchestrated
- 15ms overhead
- Hardware agnostic
3. Add new models

- Many languages, frameworks
- Instant JSON API
- Call other models seamlessly (regardless of lang)
- Granular permissions
- GPU environments
- Namespaces & versioning
Thank you!

Jon Peck Developer Advocate

jpeck@algorithmia.com
@peckjon
Algorithmia.com

FREE STUFF

$50 free at Algorithmia.com signup code: WOSC18

WE ARE HIRING

algorithmia.com/jobs

- Seattle or Remote
- Bright, collaborative env
- Unlimited PTO
- Dog-friendly