Building an Operating System for AI

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

ALGORITHMIA

Diego Oppenheimer, CEO
diego@algorithmia.com
About Me

Diego Oppenheimer - Founder and CEO - Algorithmia

- Product developer, entrepreneur, extensive background in all things data.
- Founder of algorithmic trading startup
- BS/MS Carnegie Mellon University
Make state-of-the-art algorithms discoverable and accessible to everyone.
Algorithmia.com
AI/ML scalable infrastructure on demand + marketplace

- Function-as-a-service for Machine & Deep Learning
- Discoverable, live inventory of AI
- Monetizable
- Composable
- Every developer on earth can make their app intelligent
“There’s an algorithm for that!”

70K+ DEVELOPERS  5K+ ALGORITHMS
How do we do it?

- ~5,000 algorithms  (60k w/ different versions)
- Each algorithm: 1 to 1,000 calls a second, fluctuates, no devops
- ~15ms overhead latency
- Any runtime, any architecture
Characteristics of AI

- Two distinct phases: training and inference
- Lots of processing power
- Heterogenous hardware (CPUs, GPUs, TPUs, etc.)
- Limited by compute rather than bandwidth
- “Tensorflow is open source, scaling it is not.” - Kenny Daniel
| 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.
Use Case

Jian Yang made an app to recognize food “SeeFood”. Fully trained. Works on his machine.
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.
We’ll be talking about Microservices & Serverless Computing

**MICROSERVICES**: the design of a system as independently deployable, loosely coupled services.

**ADVANTAGES**
- Maintainability
- Scalability
- Rolling deployments

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

**ADVANTAGES**
- Cost / Efficiency
- Concurrency built-in
- Speed of development
- Improved latency
Analogous to dev tool chain. Building and iterating over a model is similar to building an app.
Analogous to dev tool chain.
Building and iterating over a model is similar to building an app.

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

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

INFERENC

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

Metal or VM

Containers
**TRAINING**

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

**INFERENCE**

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

Metal or VM

Containers

Kubernetes
<table>
<thead>
<tr>
<th>TRAINING</th>
<th>OWNER: Data Scientists</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long compute cycle</td>
<td></td>
</tr>
<tr>
<td>Fixed load (Inelastic)</td>
<td></td>
</tr>
<tr>
<td>Stateful</td>
<td></td>
</tr>
<tr>
<td>Single user</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INFEERENCE</th>
<th>OWNER: DevOps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short compute bursts</td>
<td></td>
</tr>
<tr>
<td>Elastic</td>
<td></td>
</tr>
<tr>
<td>Stateless</td>
<td></td>
</tr>
<tr>
<td>Multiple users</td>
<td></td>
</tr>
</tbody>
</table>

Metal or VM

Containers

Kubernetes

REST API
Why Microservices?

- Elastic
- Scalable
- Software agnostic
- Hardware agnostic
Why Serverless?

- Cost / Efficiency
- Concurrency built-in
- 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

Max calls/s

Avg calls/s
Autoscale Architecture - Design for Local Maximum

19 machines 24 hours. $648 * 40 = $12,312 per month
Serverless Architecture - Design for Minimum

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

Portability = Low Latency
ALSO:

GPU Memory Management, Job Scheduling, Cloud Abstraction, etc.
An Operating System for AI

operating system

noun

the software that supports a computer's basic functions, such as scheduling tasks, executing applications, and controlling peripherals.
Runtime Abstraction
Support any programming language or framework, including interoperability between mixed stacks.

Elastic Scale
Prioritize and automatically optimize execution of concurrent short-lived jobs.

Cloud Abstraction
Provide portability to algorithms, including public clouds or private clouds.

Discoverability, Authentication, Instrumentation, etc.

Shell & Services
Composability

Composability is critical for AI workflows because of data processing pipelines and ensembles.
Kernel: Elastic Scale + Intelligent Orchestration

FoodClassifier
CPU util, GPU util, Memory util, IO util

FruitClassifier
CPU util, GPU util, Memory util, IO util

VeggieClassifier
CPU util, GPU util, Memory util, IO util
Kernel: Elastic Scale + 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
Kernel: Runtime Abstraction
# 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 = Algorithmia().client.file("blob://records.csv").get()
## Kernel: Cloud Abstraction

<table>
<thead>
<tr>
<th>Compute</th>
<th>EC2</th>
<th>CE</th>
<th>VM</th>
<th>Nova</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autoscaling</td>
<td>Autoscaling Group</td>
<td>Autoscaler</td>
<td>Scale Set</td>
<td>Heat Scaling Policy</td>
</tr>
<tr>
<td>Load Balancing</td>
<td>Elastic Load Balancer</td>
<td>Load Balancer</td>
<td>Load Balancer</td>
<td>LBaaS</td>
</tr>
<tr>
<td>Remote Storage</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
Summary - What makes an OS for AI?

- Stack-agnostic
- Composable
- Self-optimizing
- Auto-scaling
- Monitorable
- Discoverability
Unix
Multi-tenancy, Composability

GUI (Win/Mac)
Accessibility

Punched Cards
1970s

DOS
Hardware Abstraction

iOS/Android
Built-in App Store (Discoverability)
AI is here

Punched Cards
1970s

iOS/Android
Built-in App Store
(Discoverability)
FREE STUFF:

Signup with code: NVIDIA-GTC18 for $50 on us.

Thank you!

Diego Oppenheimer
CEO

diego@algorithmia.com
@doppenhe
more slides
The New Moats

Source: Jerry Chen, Greylock Ventures
# init
client = Algorithmia.client()

# get data (S3)
s3 = boto3.client("s3")
obj = s3.get_object(Bucket="bucket-name", Key="records.csv")
data = obj["Body"].read()

# remove seasonality
data = client.algo("ts/RemoveSeasonality").pipe(data).result

# forecast time series
data = client.algo("ts/ForecastLSTM").pipe(data).result

# init
client = Algorithmia.client()

# get data (anything)
data = client.file("blob://records.csv").get()

# remove seasonality
data = client.algo("ts/RemoveSeasonality").pipe(data).result

# forecast time series
data = client.algo("ts/ForecastLSTM").pipe(data).result
# MY_ALGORITHM.py

client = Algorithmia.client()
data = client.file("blob://records.csv").get()

# remove seasonality
data = client.algo("ts/RemoveSeasonality").pipe(data).result

# forecast time series
data = client.algo("ts/ForecastLSTM").pipe(data).result
Kernel: Elastic Scale + Intelligent Orchestration

```python
# MY_ALGORITHM.py

client = Algorithmia.client()
data = client.file("blob://records.csv").get()

# remove seasonality
data = client.algo("ts/RemoveSeasonality").pipe(data).result

# forecast time series
data = client.algo("ts/ForecastLSTM").pipe(data).result
```
Kernel: Elastic Scale + 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
Kernel: Runtime Abstraction

```python
# MY_ALGORITHM.py

client = Algorithmia.client()
data = client.file("blob://records.csv").get()

# remove seasonality
data = client.algo("ts/RemoveSeasonality").pipe(data).result

# forecast time series
data = client.algo("ts/ForecastLSTM").pipe(data).result
```
Challenges

- **Machine learning**
  - CPU/GPU/Specialized hardware
  - Multiple frameworks, languages, dependencies
  - Called from different devices/architectures

- **“Snowflake” environments**
  - Unique cloud hardware and services

- **Uncharted territory**
  - Not a lot of literature, errors messages sometimes cryptic (can’t just stackoverflow)