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

Jon Peck

Full-Spectrum Developer & Advocate

ALGORITHMIA

jpeck@algorithmia.com
@peckjon
Algorithmia.com

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

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**

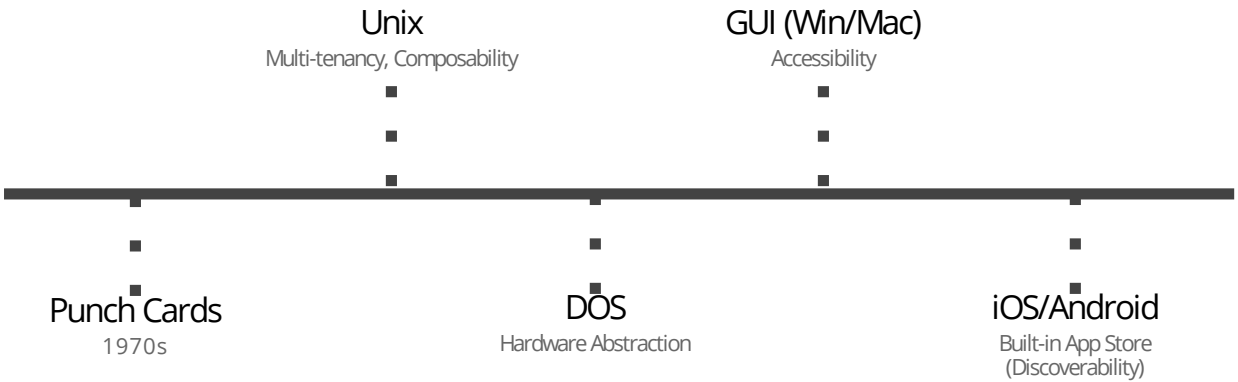
An Operating System for AI

What did the evolution of OS look like?

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

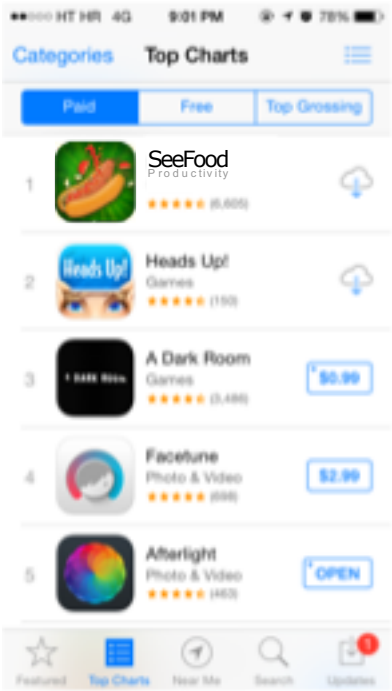
Jian Yang made an app to recognize food “SeeFood”

Use Case

He deployed his trained model to a GPU-enabled server

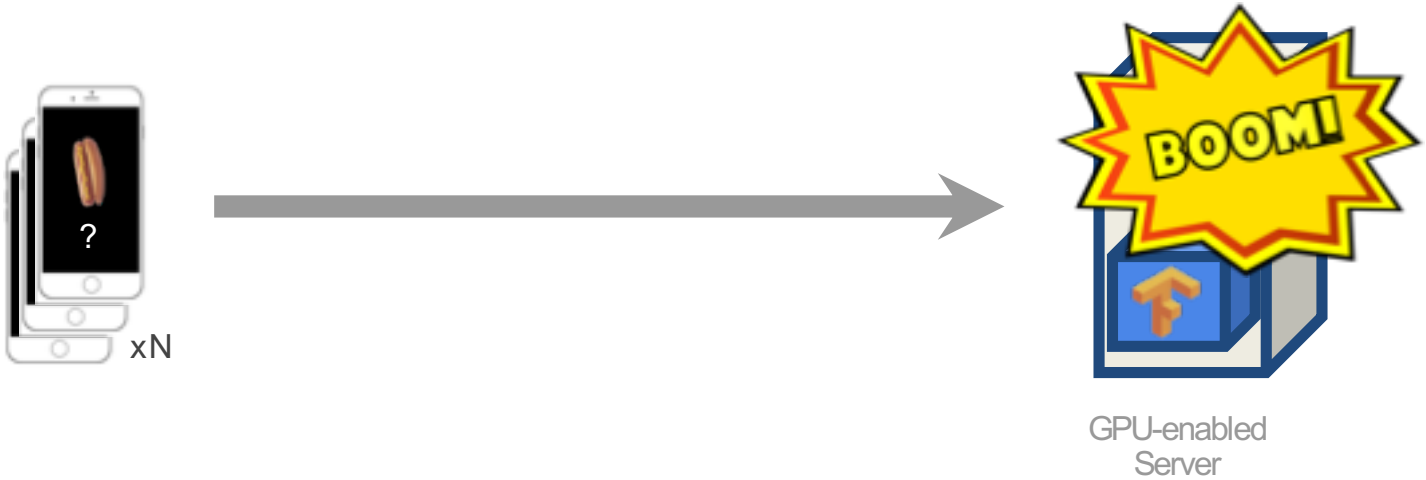
Use Case

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

TRAINING

OWNER: Data Scientists

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

INFERENCE

OWNER: DevOps

- Short compute bursts
- Stateless
- Elastic
- Multiple users

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

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.

Metal or VM

INFERENCE

OWNER: DevOps

- Short compute bursts
- Stateless
- Elastic
- Multiple users

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

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

INFERENCE

OWNER: DevOps

- ☑ Short compute bursts
- ☑ Stateless
- ☐ Elastic
- ☐ Multiple users

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

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

INFERENCE

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

INFERENCE

OWNER: DevOps

- ☑ Short compute bursts
- ☑ Stateless
- ☑ Elastic
- ☑ Multiple users

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

Containers

Kubernetes

Microservices & Serverless Computing => ML Hosting

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

ADVANTAGES

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

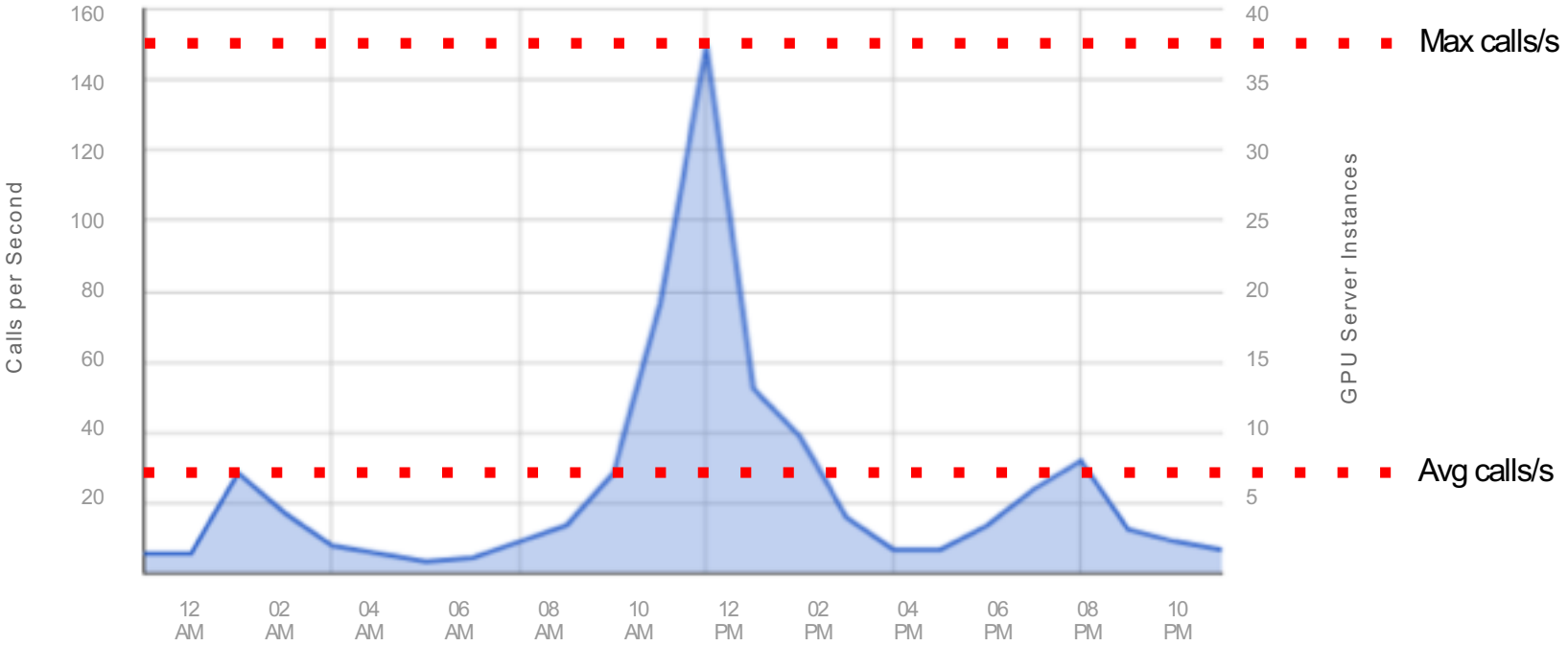
SERVERLESS: the encapsulation, starting, and stopping of singular functions per request, with a just-in-time-compute model.

ADVANTAGES

- Elasticity, Cost Efficiency
- Concurrency
- Improved Latency

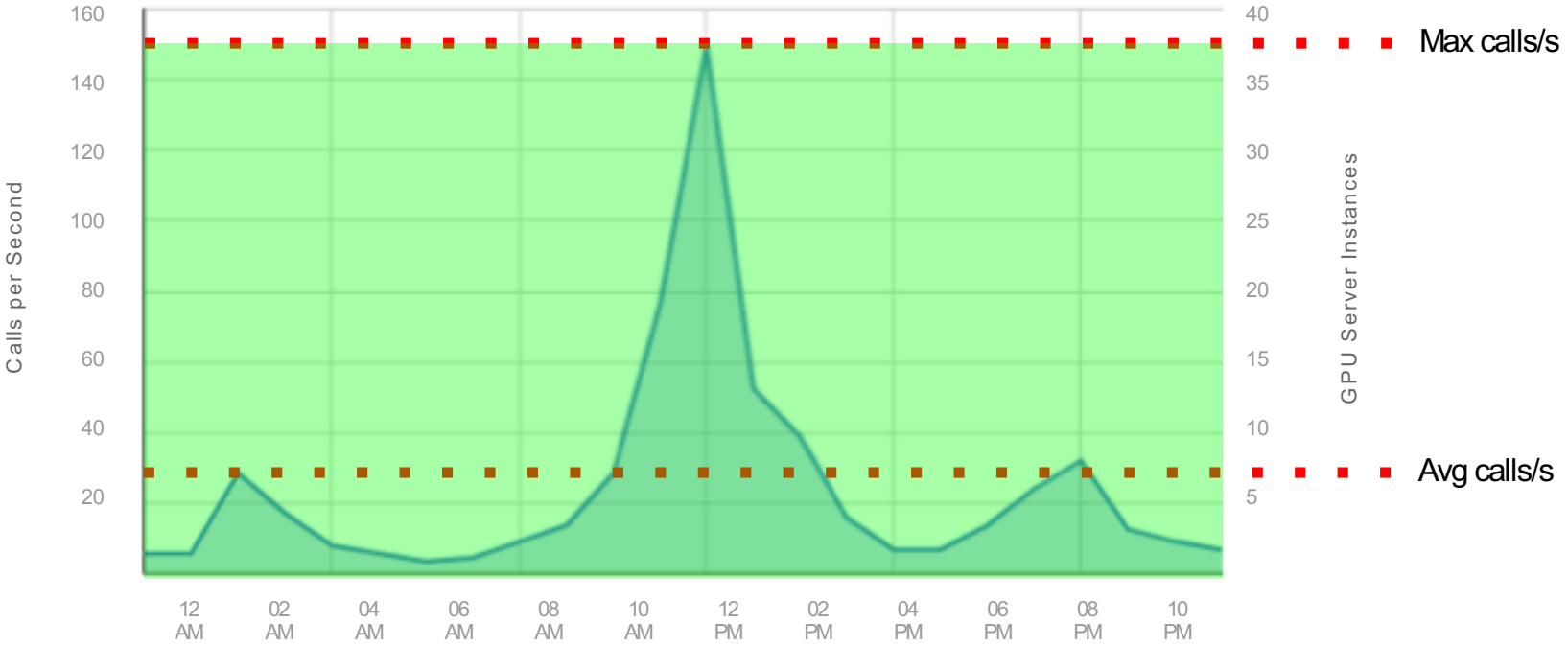
Why Serverless - Cost Efficiency

Jian Yang's "SeeFood" is most active during lunchtime.



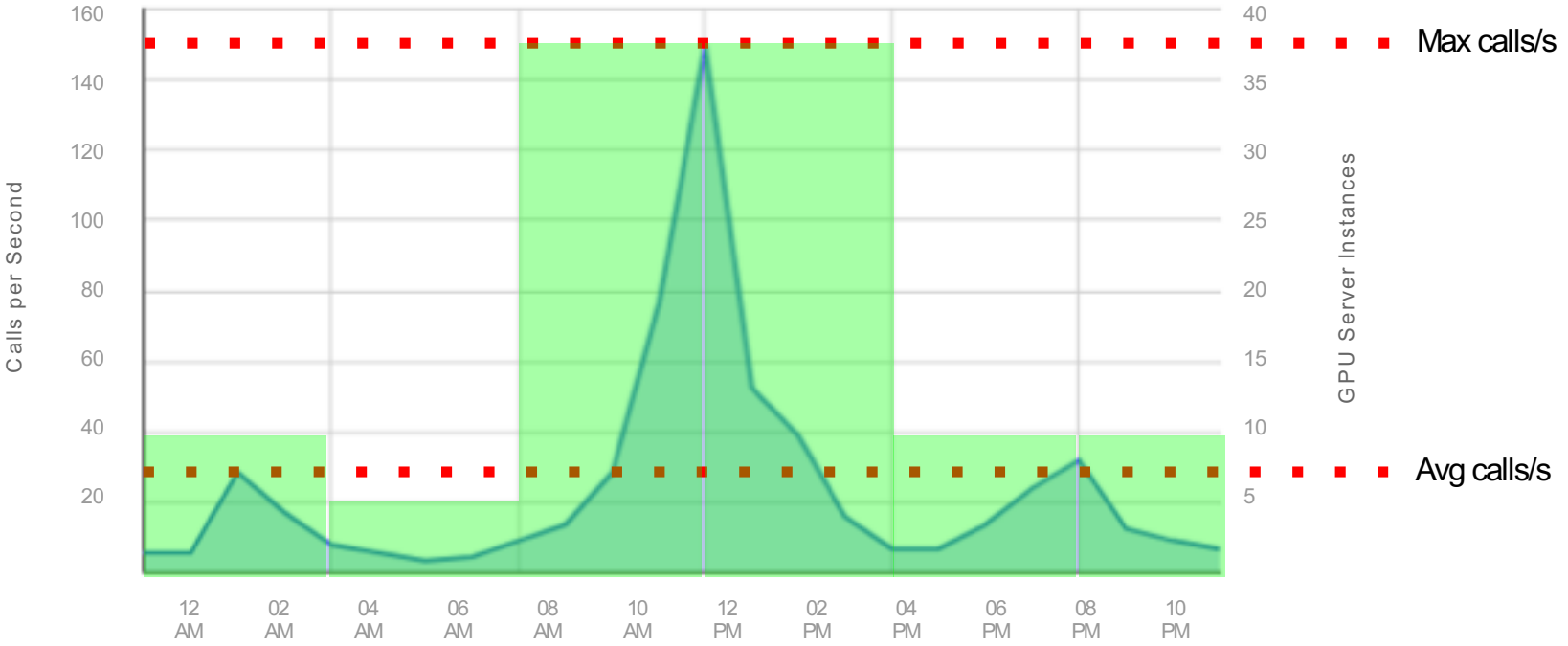
Traditional Architecture - Design for Maximum

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



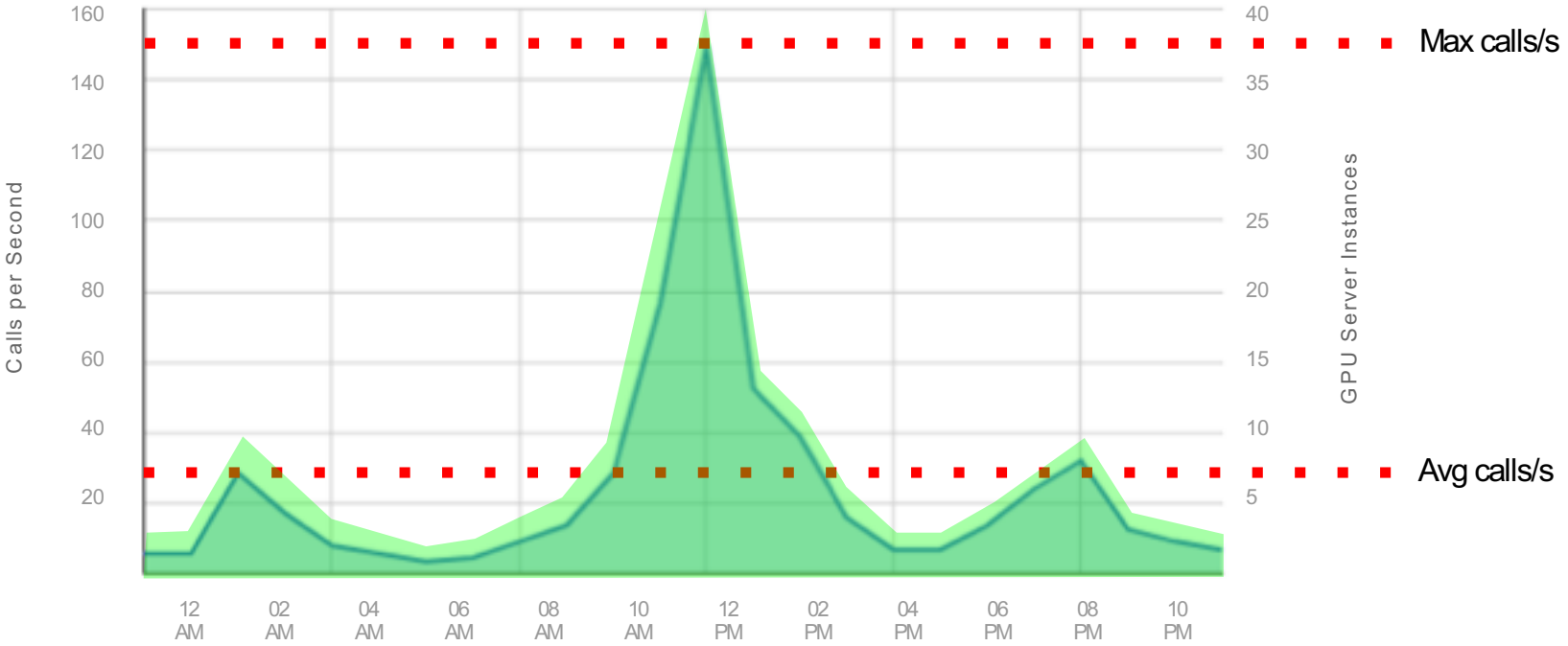
Autoscale Architecture - Design for Local Maximum

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

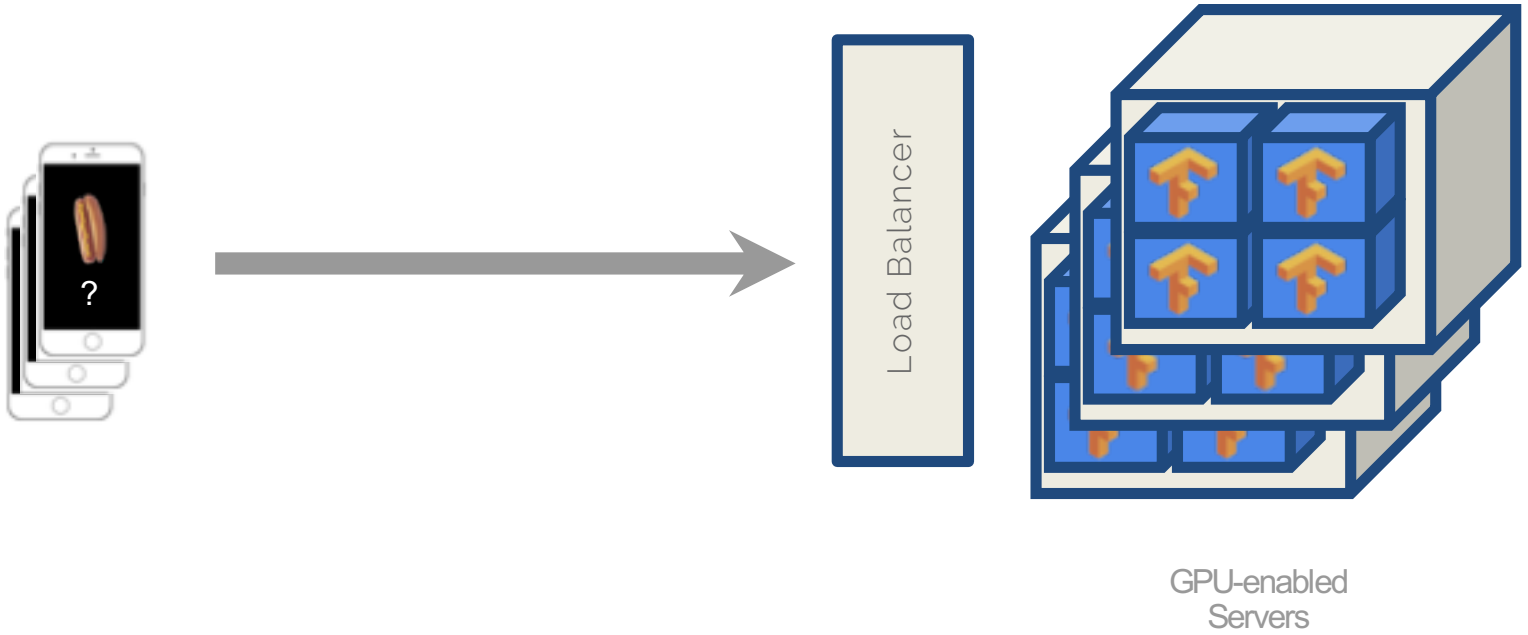


Serverless Architecture - Design for Minimum

Avg. of 21 calls / sec, or equivalent of 6 machines. $648 * 6 = **$3,888 per month**



Why Serverless - Concurrency



Why Serverless - Improved Latency

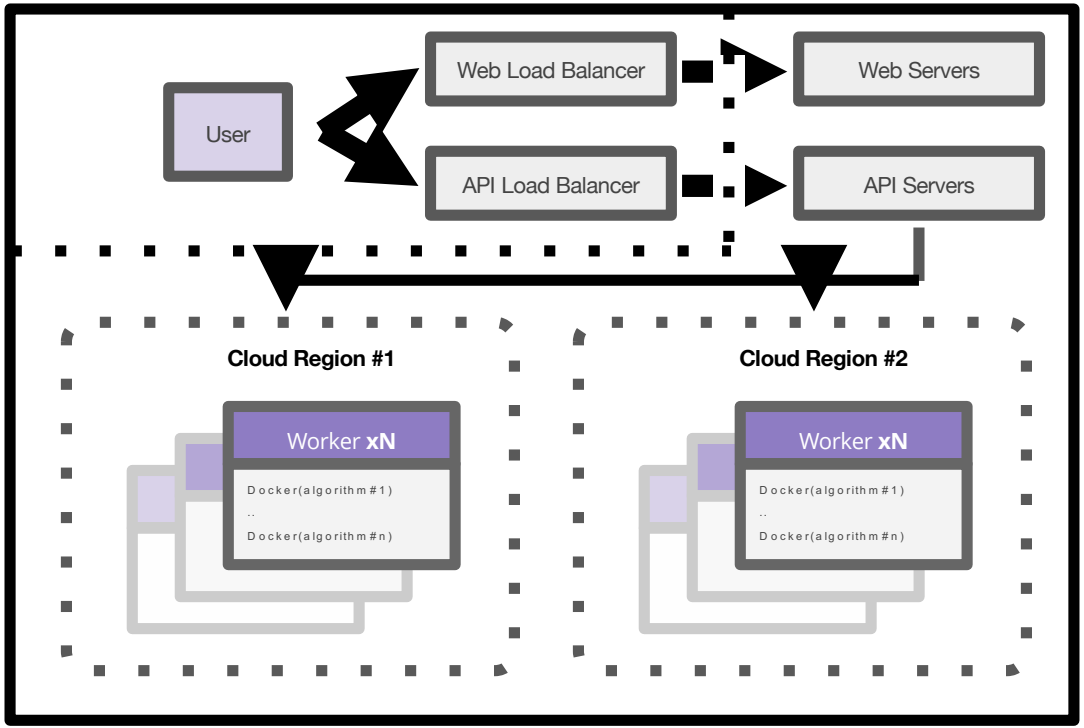
Portability = Low Latency

Almost there! We also need:

GPU Memory Management, Job Scheduling, Cloud Abstraction,

Discoverability, Authentication, Logging, etc.

Elastic Scale



Elastic Scaling with Intelligent Orchestration

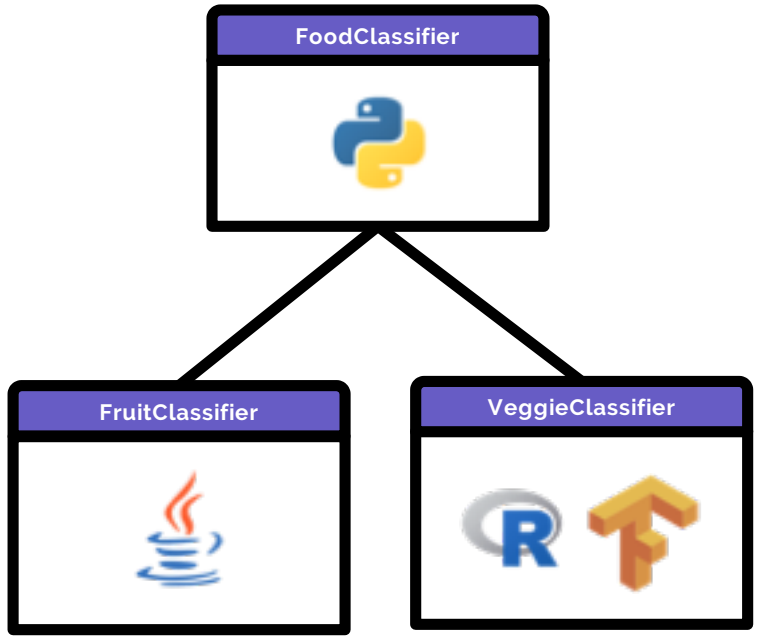
Knowing that:

- **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

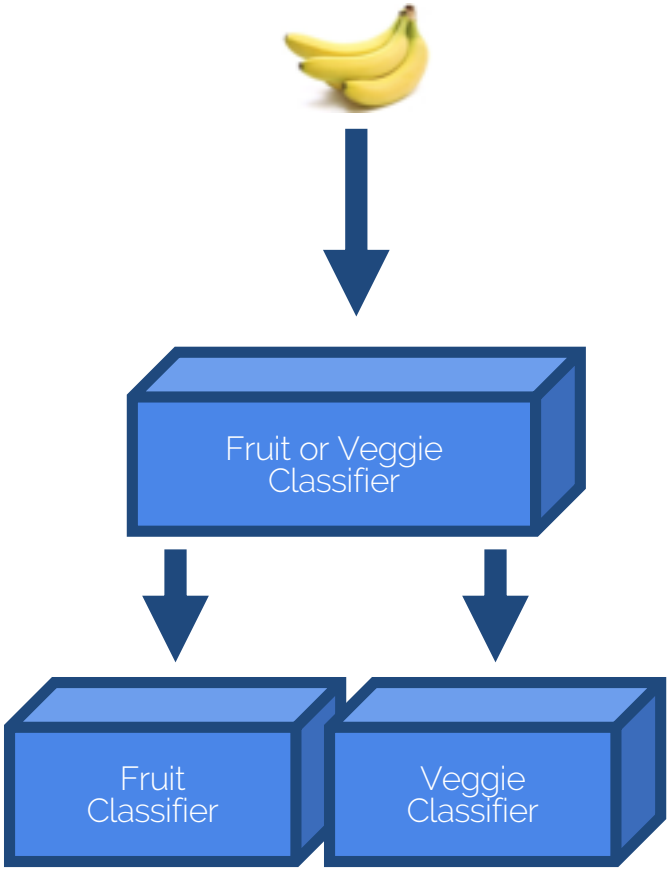
Runtime Abstraction



Composability

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

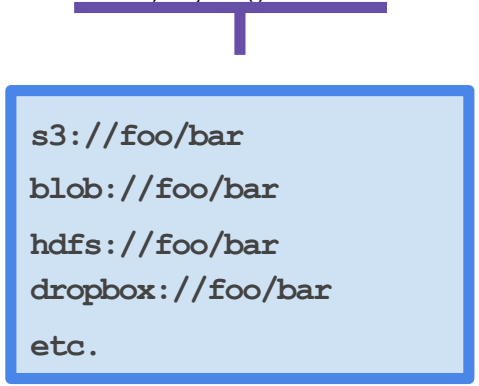
```
cat file.csv | grep foo | wc -l
```



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()
```

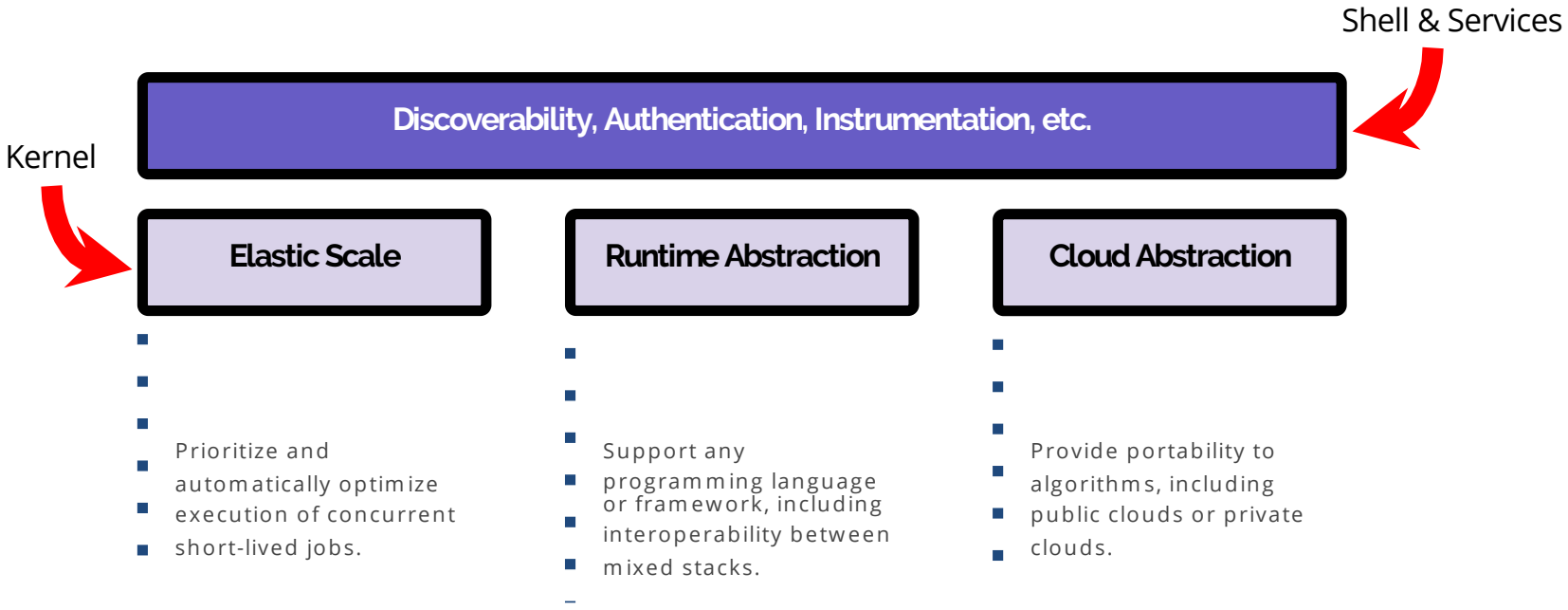


Cloud Abstraction

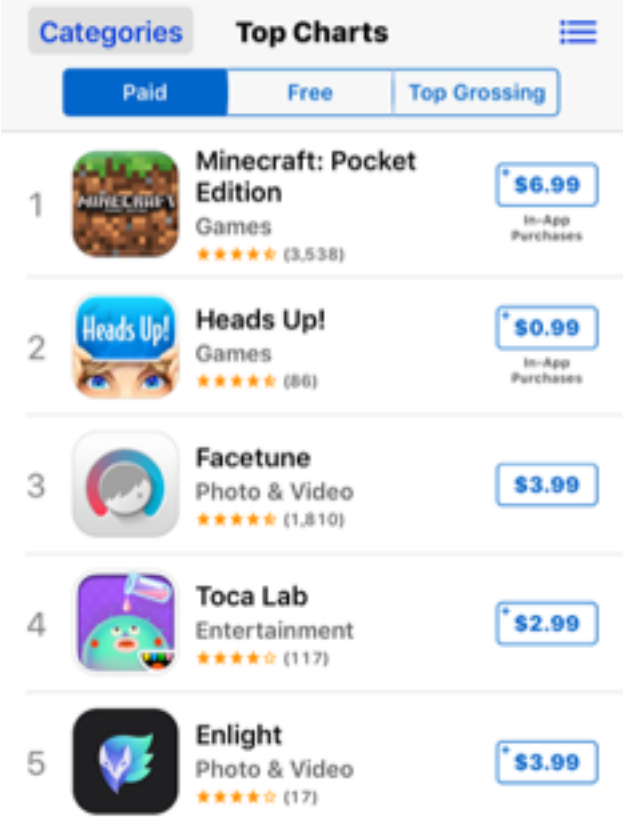
				
Compute	EC2	CE	VM	Nova
Autoscaling	Autoscaling Group	Autoscaler	Scale Set	Heat Scaling Policy
Load Balancing	Elastic Load Balancer	Load Balancer	Load Balancer	LBaaS
Remote Storage	Elastic Block Store	Persistent Disk	File Storage	Block Storage

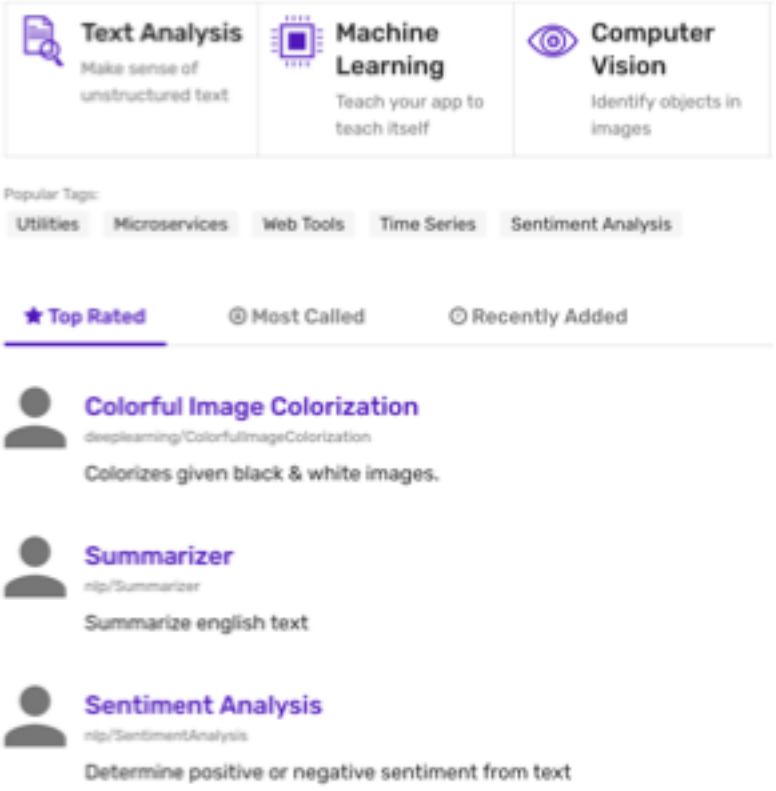
Partial Source: Sam Ghods, KubeConf 2016

An Operating System for AI: the “AI Layer”



Discoverability: an App Store for AI



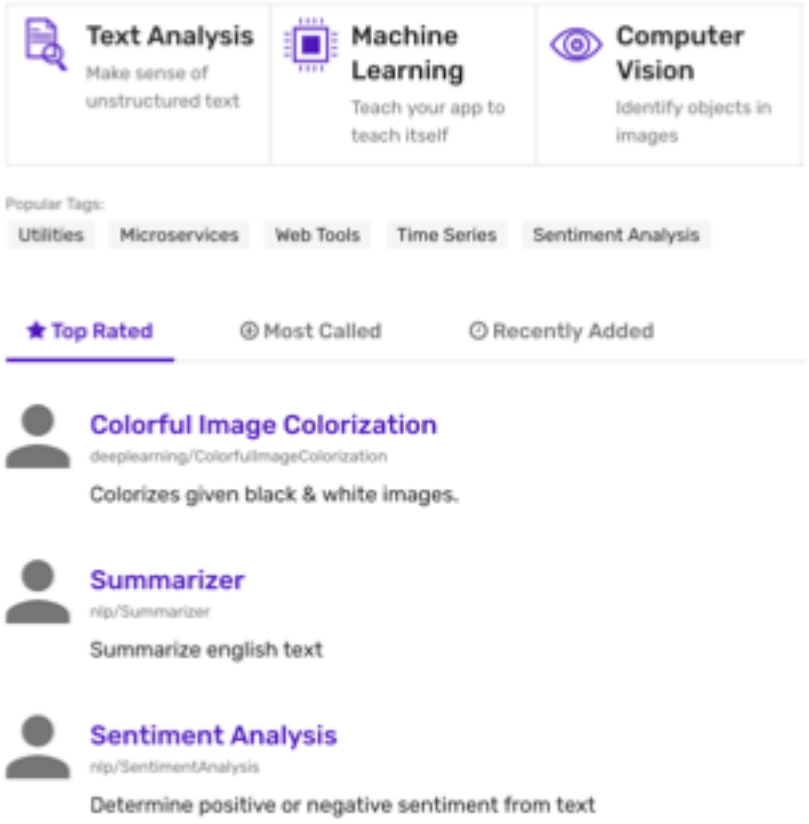


Algorithmia's OS for AI: *discover* a model

1. Discover a model

- AppStore-like interface
- Categorized, tagged, rated
- Well-described

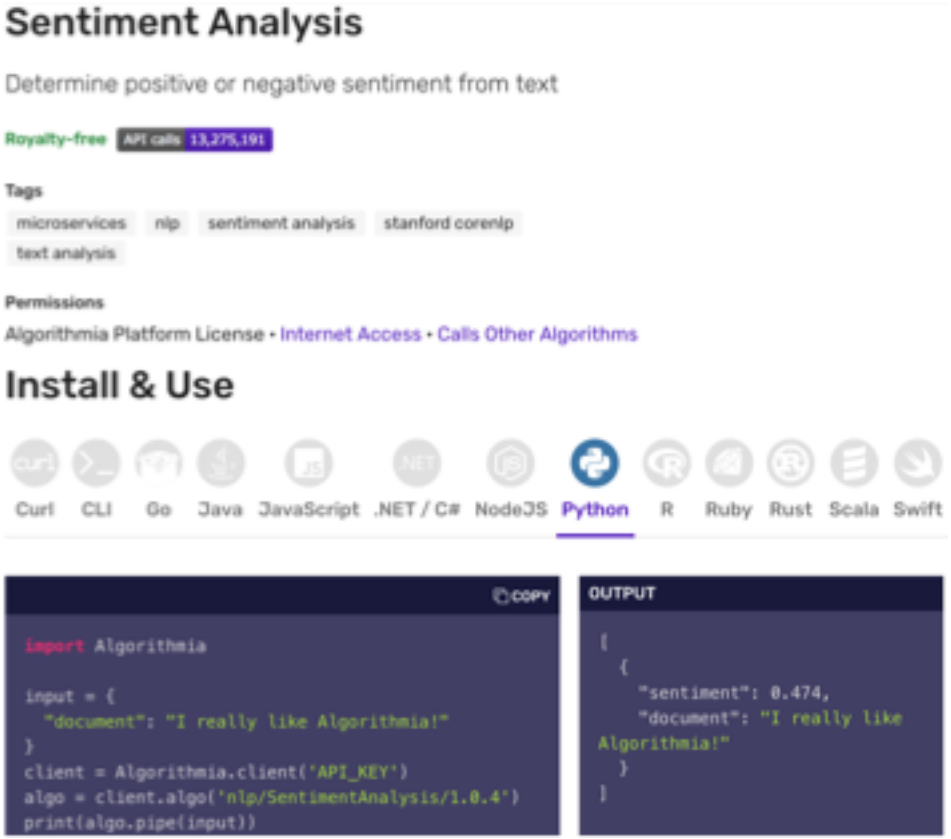
  (purpose, source, API)



Algorithmia's OS for AI: *execute* a model

2. Execute from any language

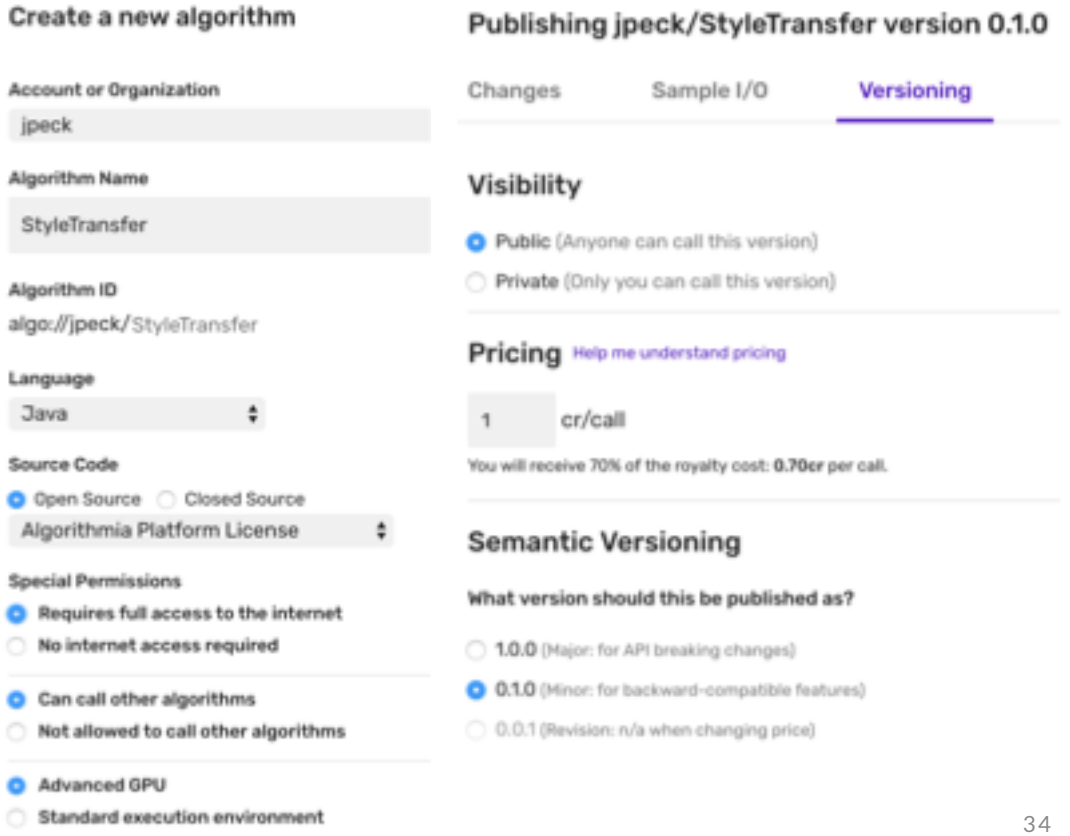
- Raw JSON, or lang stubs
- Common syntax
- Autoscaled elastic cloud-exec
- Secure, isolated
- Concurrent, orchestrated
- 15ms overhead
- Hardware agnostic



Algorithmia's OS for AI: *add* a model

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

ALGORITHMIA

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**