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.
- Microsoft: PowerPivot, PowerBI, Excel and SQL Server.
- Founder of algorithmic trading startup
- BS/MS Carnegie Mellon University

ALGORITHMIA

Make state-of-the-art algorithms

discoverable and **accessible**

to everyone.

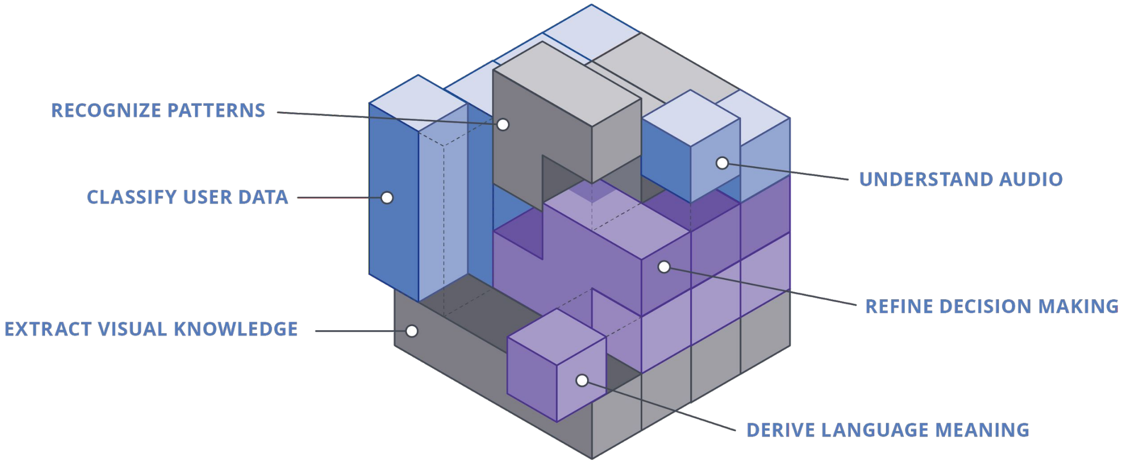
Algorithmia.com

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!”

63K DEVELOPERS   4.8K ALGORITHMS



How do we do it?

- ~4,800 algorithms (50k 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

TRAINING

OWNER: Data Scientists

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

INFERENCE

OWNER: DevOps

- [ ] Short compute bursts
- [ ] Elastic
- [ ] Stateless
- [ ] Multiple users

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

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

INFERENCE

OWNER: DevOps

- [ ] Short compute bursts
- [ ] Elastic
- [ ] Stateless
- [ ] Multiple users

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

TRAINING

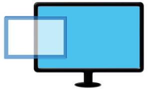
OWNER: Data Scientists

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

INFERENCE

OWNER: DevOps

- Short compute bursts
- Elastic
- Stateless
- Multiple users



Metal or VM

Containers

TRAINING

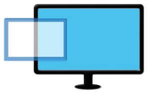
OWNER: Data Scientists

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

INFERENCE

OWNER: DevOps

- [x] Short compute bursts
- [x] Elastic
- [x] Stateless
- [ ] Multiple users



Metal or VM

Containers

Kubernetes

TRAINING

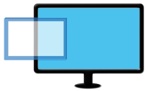
OWNER: Data Scientists

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

INFERENCE

OWNER: DevOps

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



Metal or VM

Containers

Kubernetes

REST API

Why Microservices?

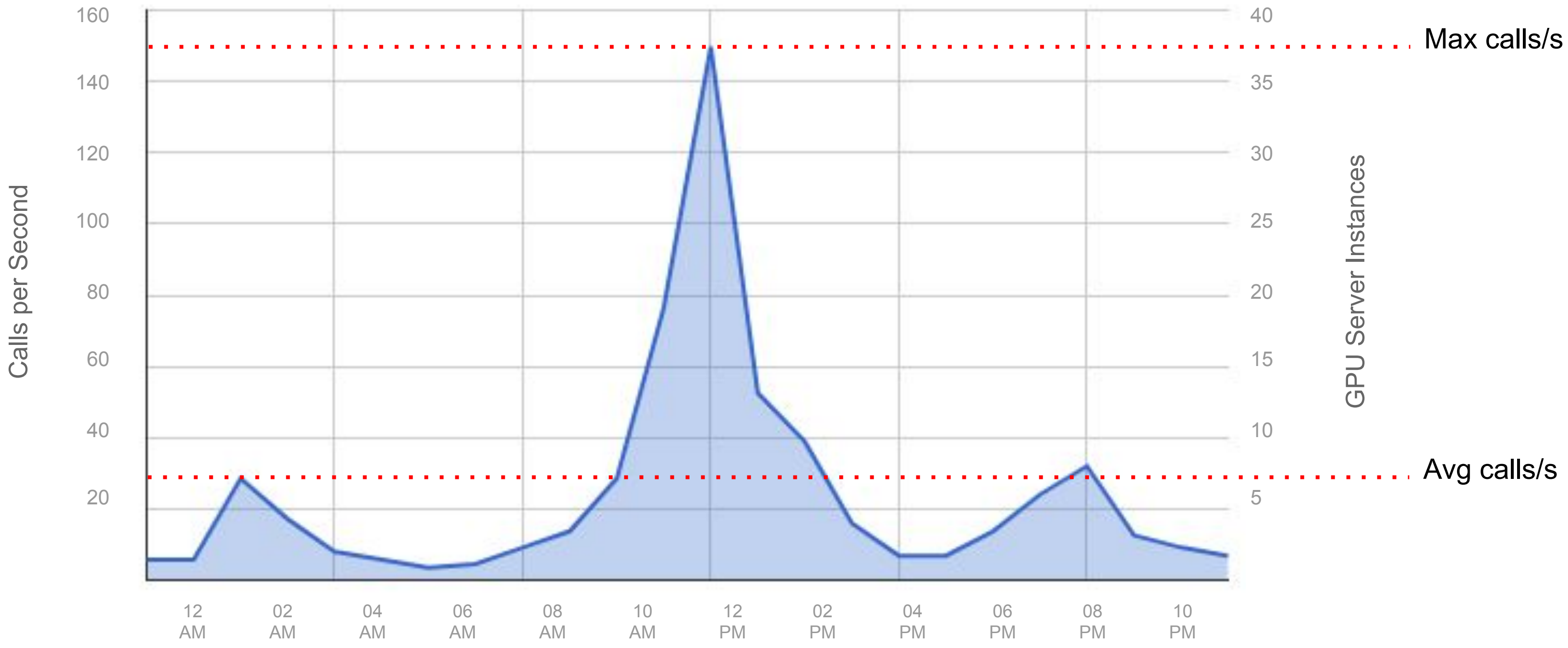
- Elastic
- Scalable
- Software agnostic
- Hardware agnostic

Why Serverless?

- Cost / Efficiency
- Concurrency built-in
- Improved latency

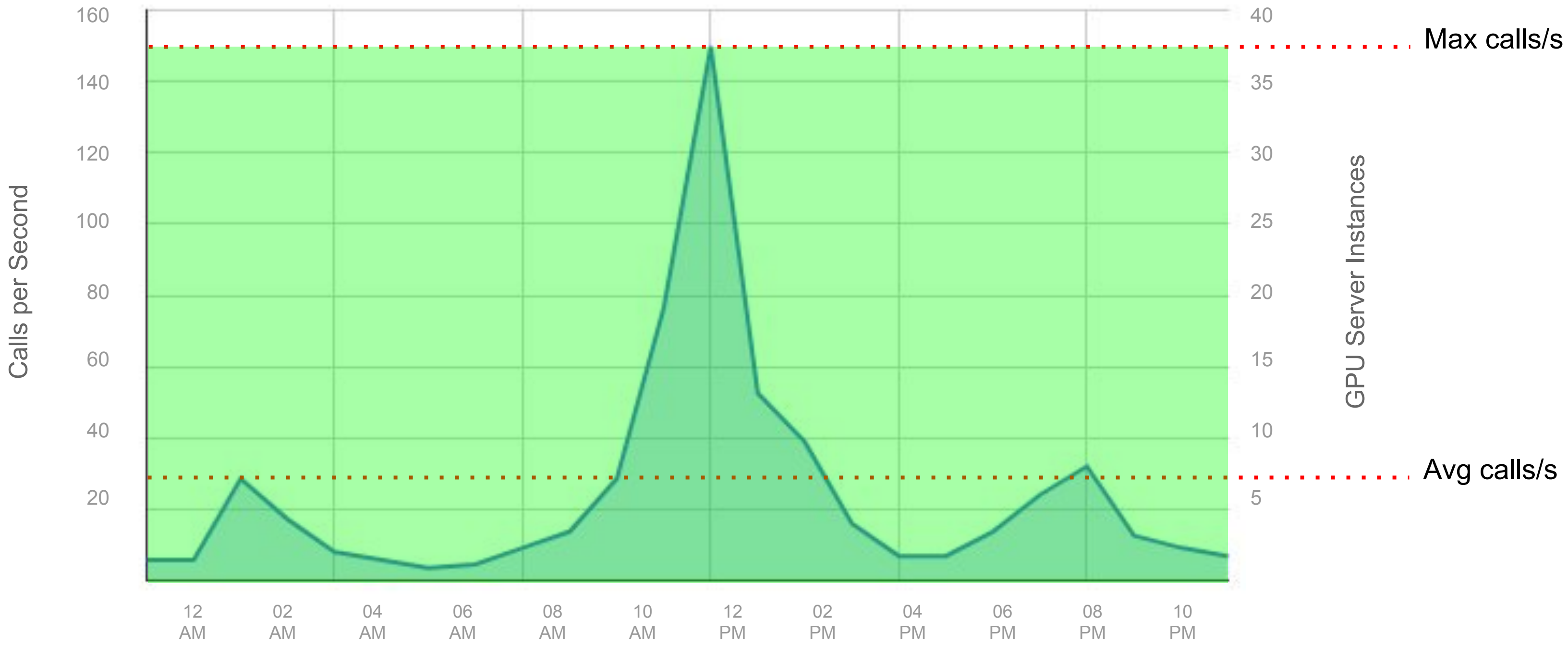
Why Serverless - Cost Efficiency

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



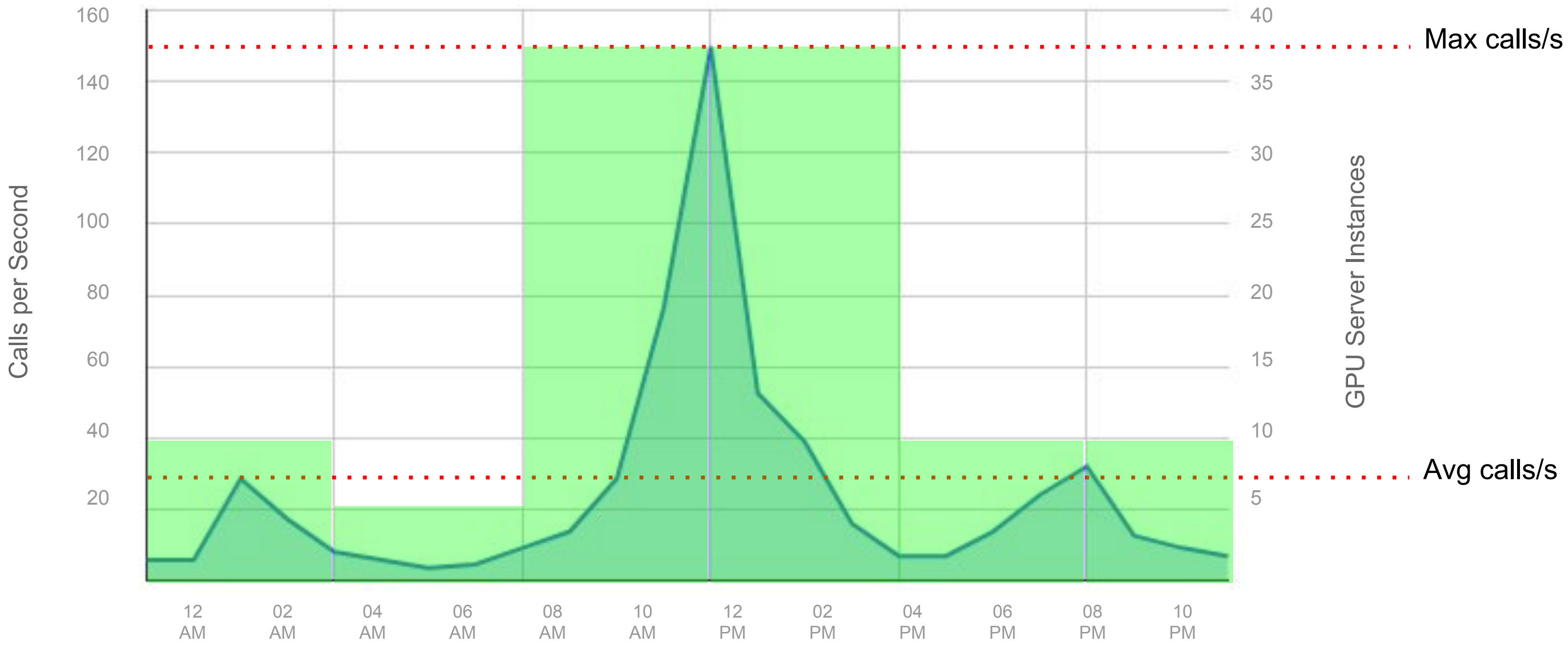
Traditional Architecture - Design for Maximum

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



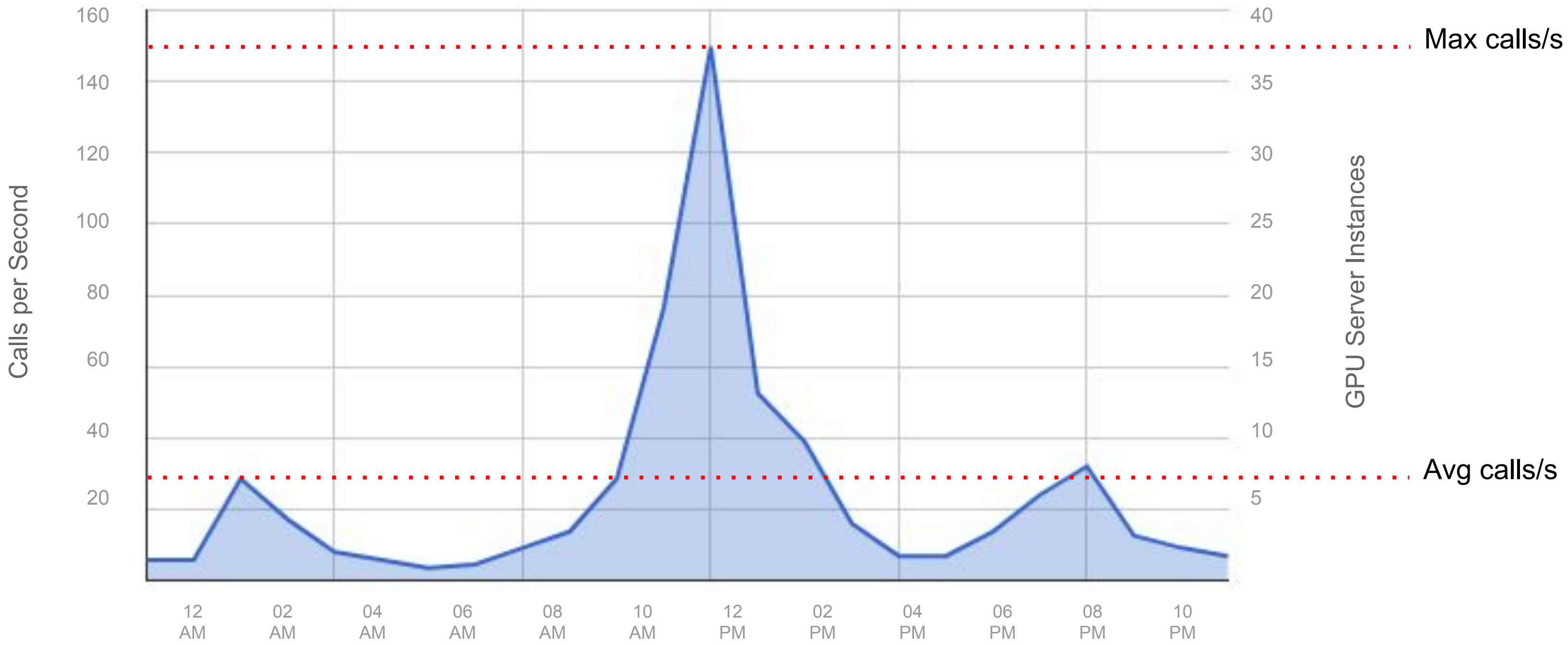
Autoscale Architecture - Design for Local Maximum

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

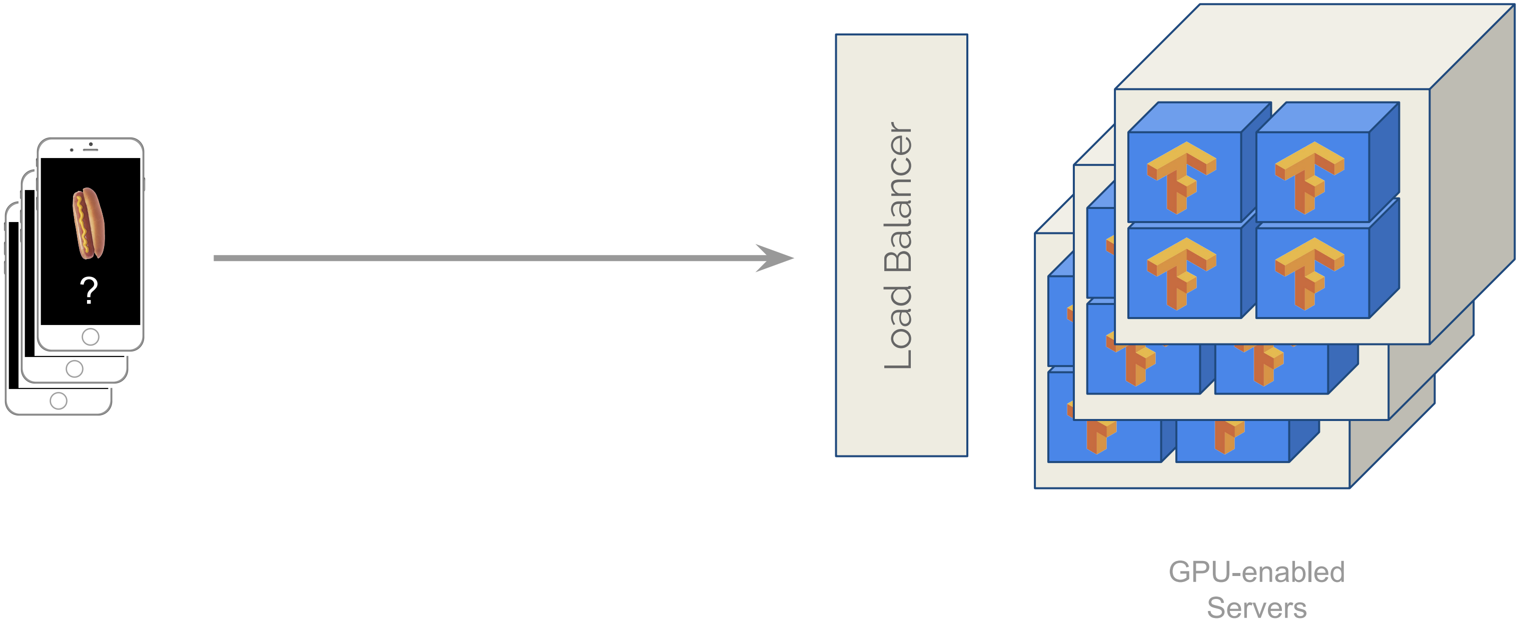


Serverless Architecture - Design for Minimum

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

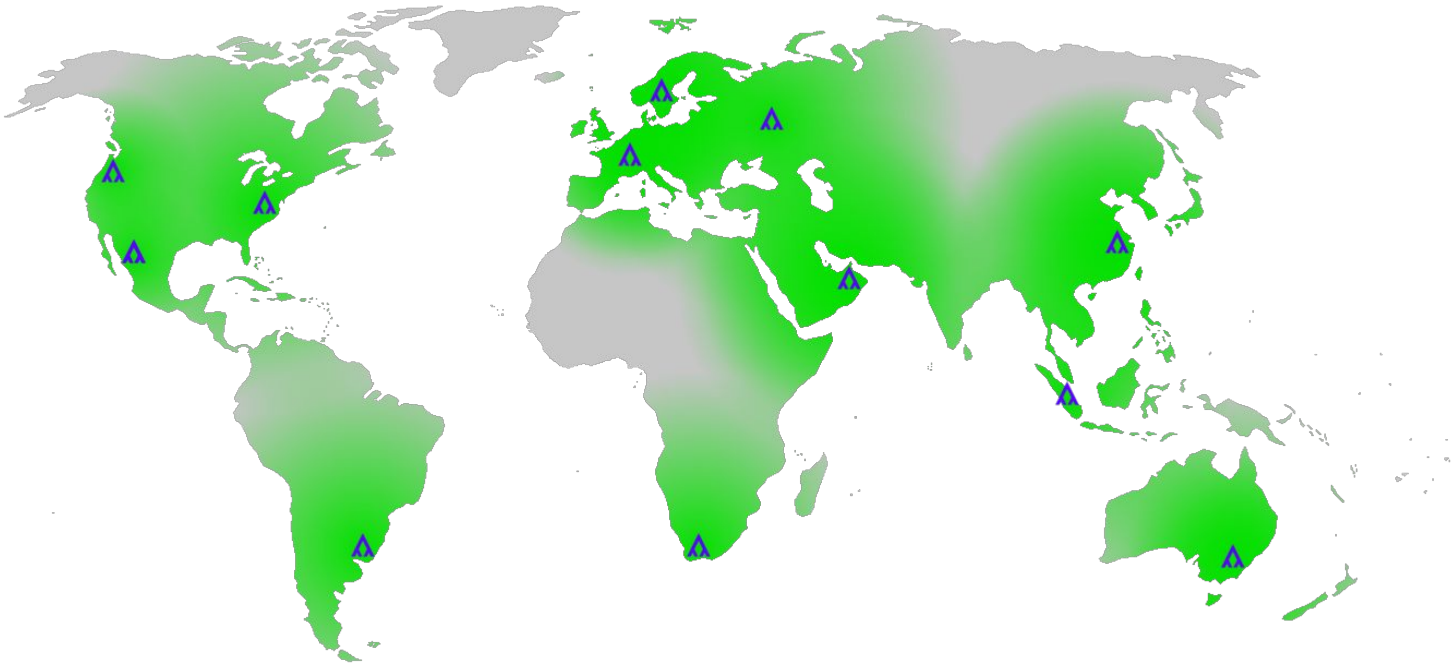


Why Serverless - Concurrency



Why Serverless - Improved Latency

Portability = Low Latency



+ +

ALSO:

GPU Memory Management, Job Scheduling, Cloud Abstraction, etc.

An Operating System for AI

op·er·at·ing sys·tem

/ˈäpəˌrādiNG ˌsistəm/

noun

the software that supports a computer's basic functions, such as scheduling tasks, executing applications, and controlling peripherals.

Translations, word origin, and more definitions

Shell & Services

Discoverability, Authentication, Instrumentation, etc.

Kernel

Elastic Scale

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

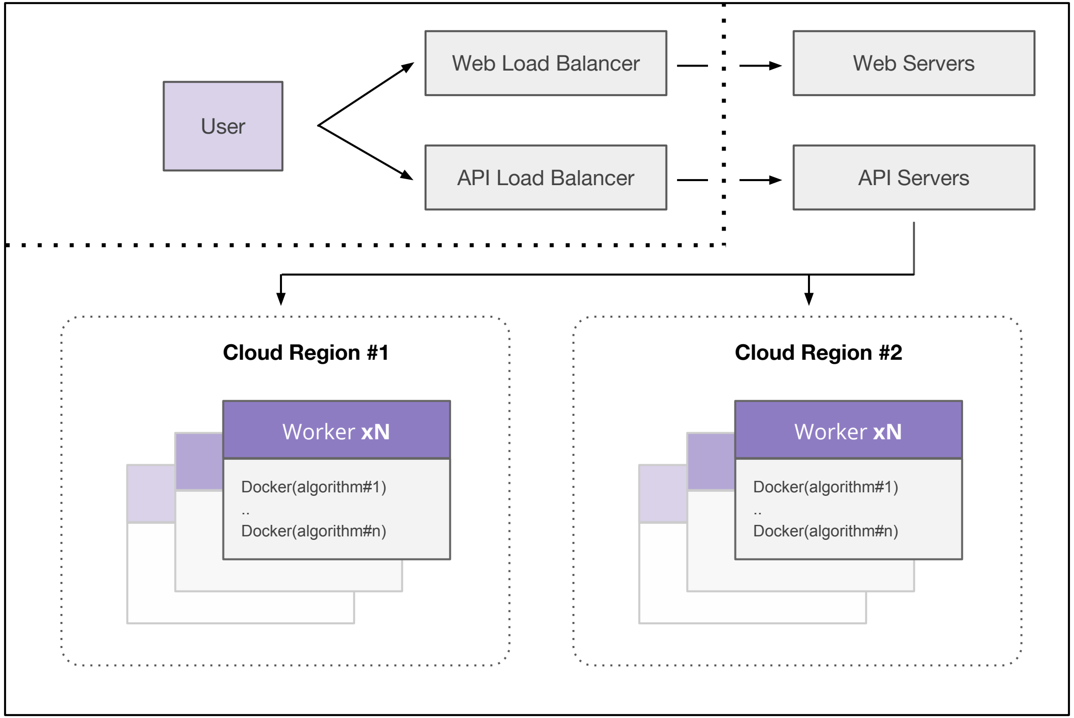
Runtime Abstraction

Support any programming language or framework, including interoperability between mixed stacks.

Cloud Abstraction

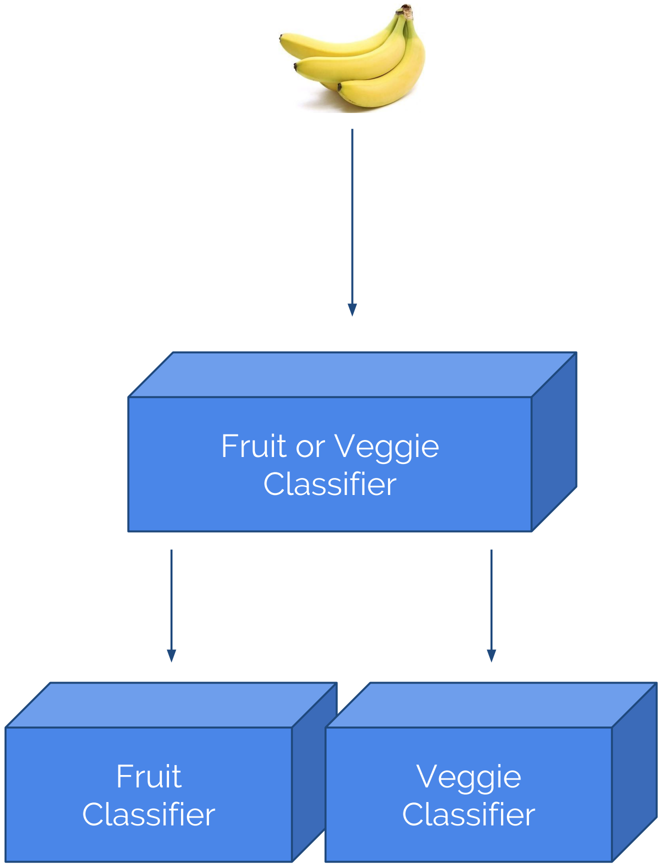
Provide portability to algorithms, including public clouds or private clouds.

Kernel: Elastic Scale

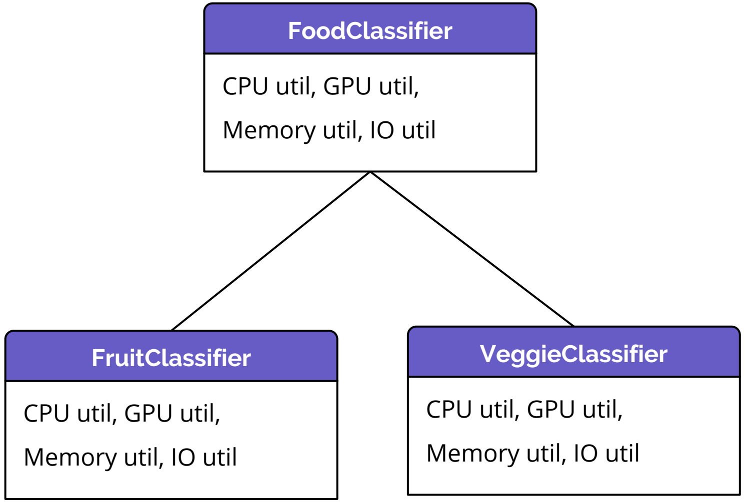


Composability

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



Kernel: Elastic Scale + Intelligent Orchestration



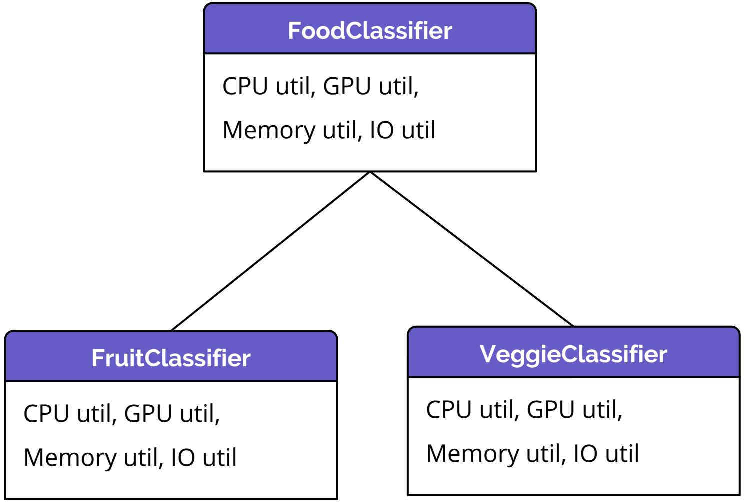
Kernel: Elastic Scale + Intelligent Orchestration

Knowing that:

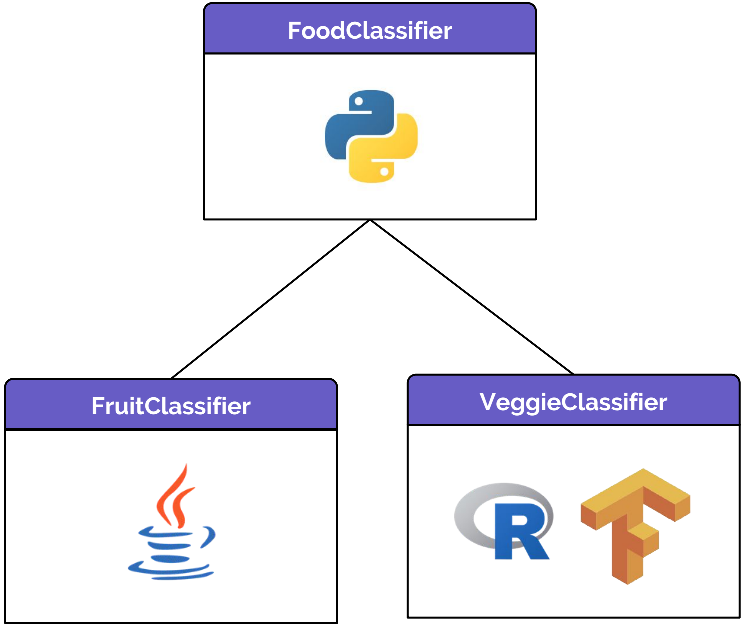
- **Algorithm A** always calls **Algorithm B**
- Algorithm A consumes X CPU, X Memory, etc
- Algorithm B consumes X CPU, X Memory, etc

Therefore we can slot them in a way that:

- Reduce network latency
- Increase cluster utilization
- Build dependency graphs



Kernel: Runtime Abstraction



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

```
s3://foo/bar
blob://foo/bar
hdfs://foo/bar
dropbox://foo/bar
etc.
```

Kernel: Cloud Abstraction

	amazon web services	Google Cloud Platform	Microsoft Azure	openstack
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

Summary - What makes an OS for AI?

Stack-agnostic

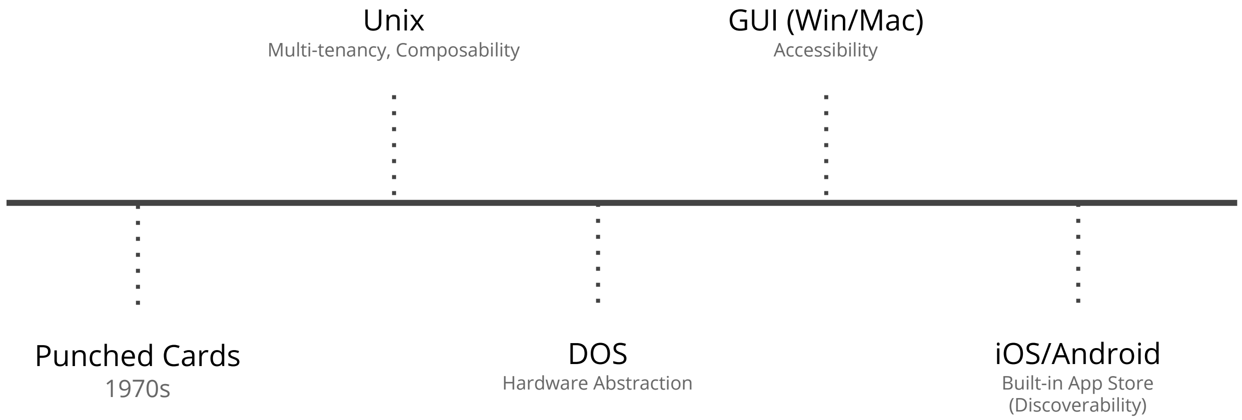
Composable

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:
CloudSummit17 for $50 on us.

ALGORITHMIA

Thank you!

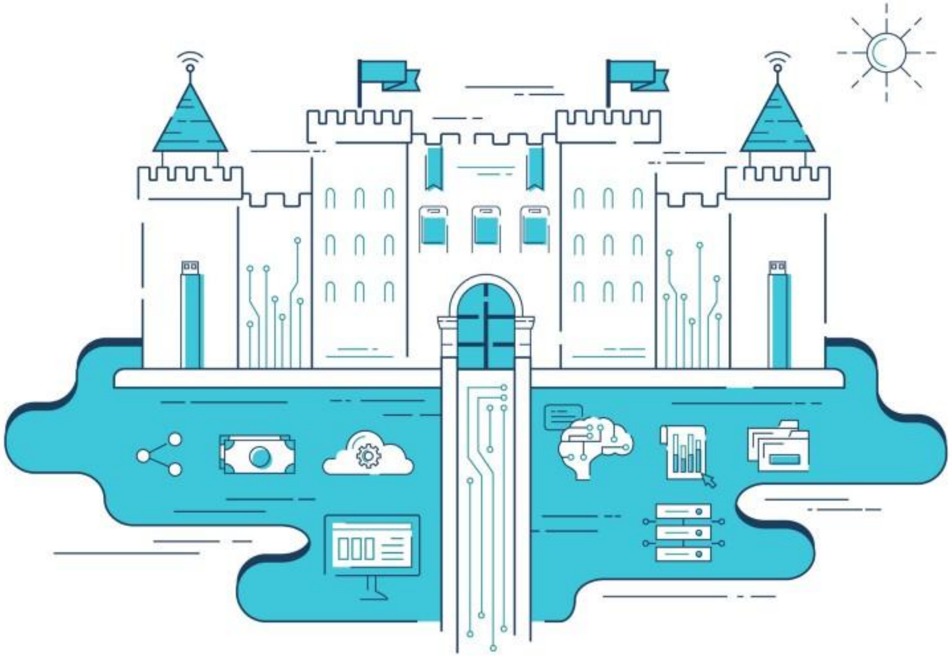
Diego Oppenheimer
CEO

diego@algorithmia.com

@doppenhe

more slides

The New Moats



Source: Jerry Chen, Greylock Ventures


Punched Cards
1970s
GitHub and Heroku
Today

Kernel: Cloud Abstraction - Storage

```
# 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
```

```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
```

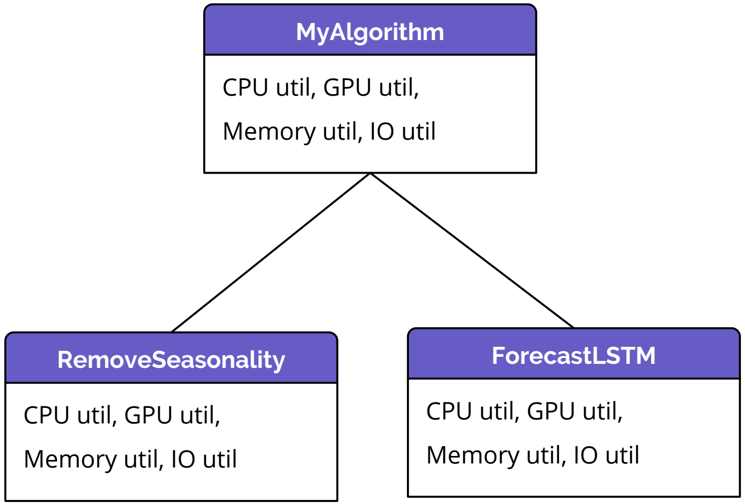
Kernel: Elastic Scale + Intelligent Orchestration

```
# 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
```



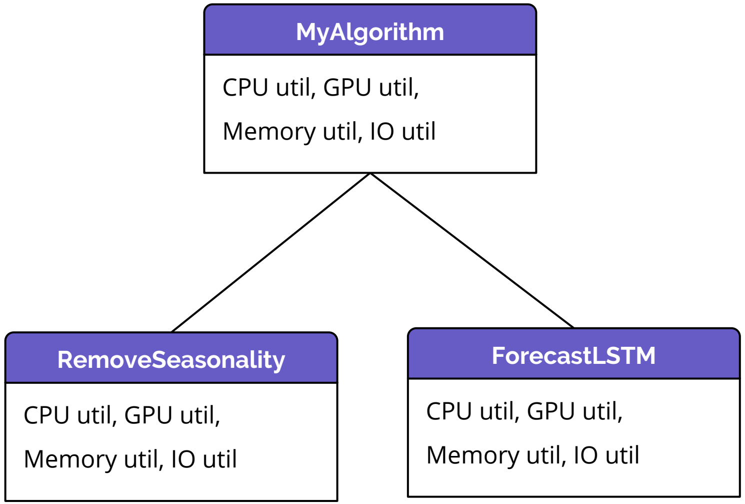
Kernel: Elastic Scale + Intelligent Orchestration

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



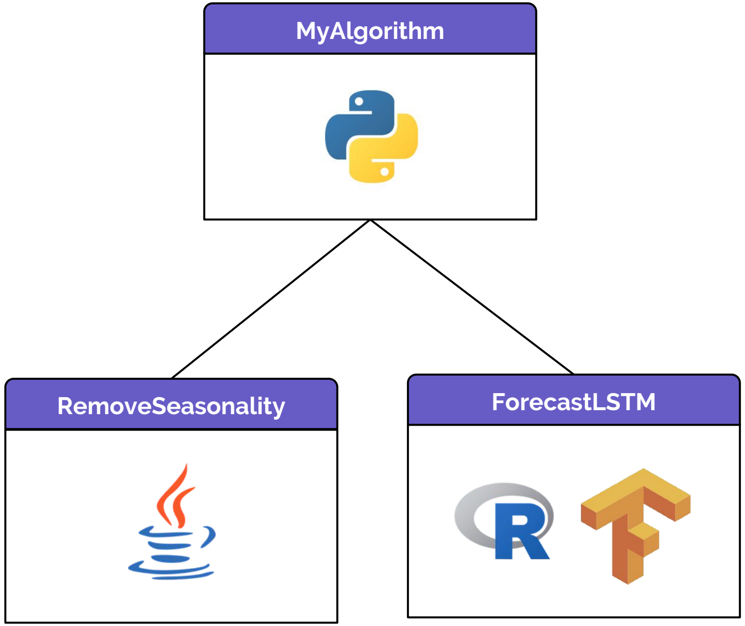
Kernel: Runtime Abstraction

```
# 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)