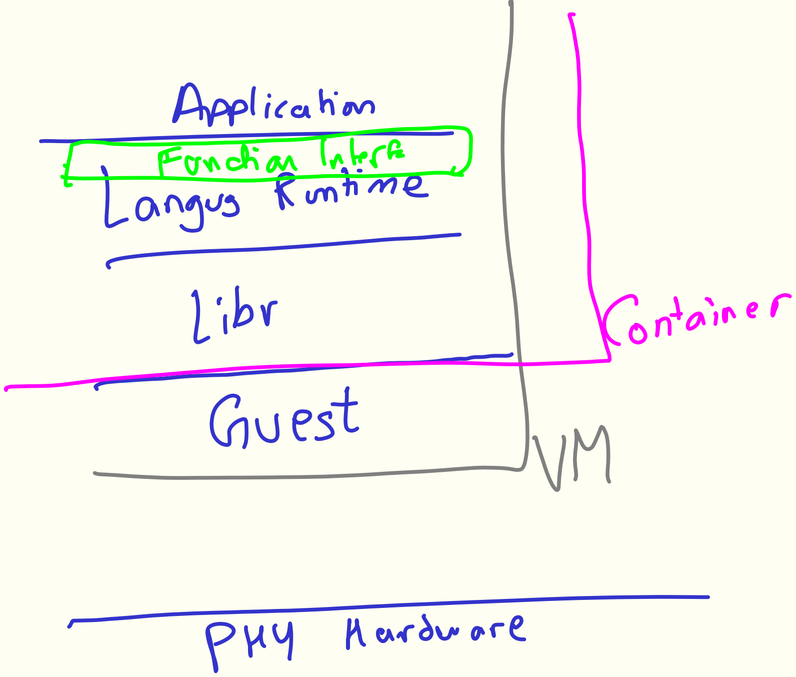
Functions as a Service

Infrastructure as a Service Pitfalls

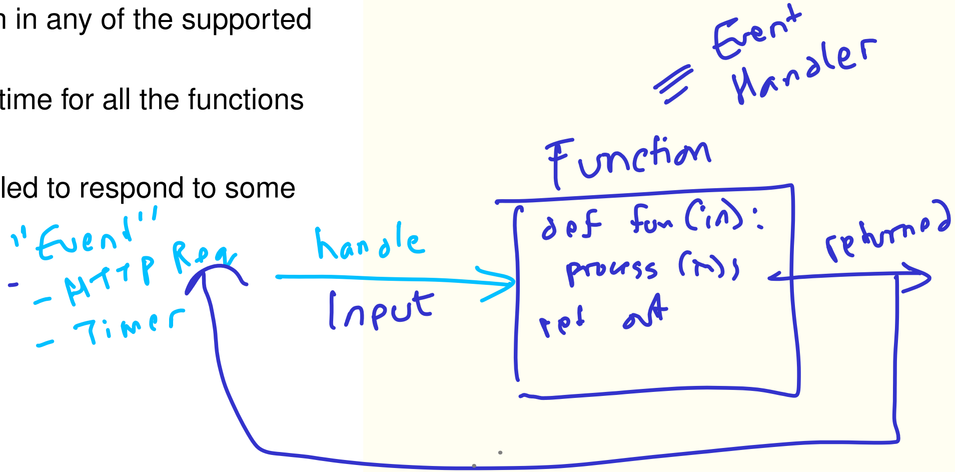
Infrastructure is now software, but even that is too hard...

- Configuring and managing complete software stack
- OS and software upgrades, security patches, . . .
- Monitoring and logging
- Auto-scaling, redundancy, geo-replication, . . .



Cloud Functions

- Cloud platform runs *functions* on behalf of user
- `ret-val function-name(arguments) {...}`
- Users provide the function implementation in any of the supported languages
- Cloud platform provides the language runtime for all the functions
- Python, Javascript, Java, Go, . . .
- Event-driven programming: function is called to respond to some event



```
def fun(in):
    process(in);
ret out
```

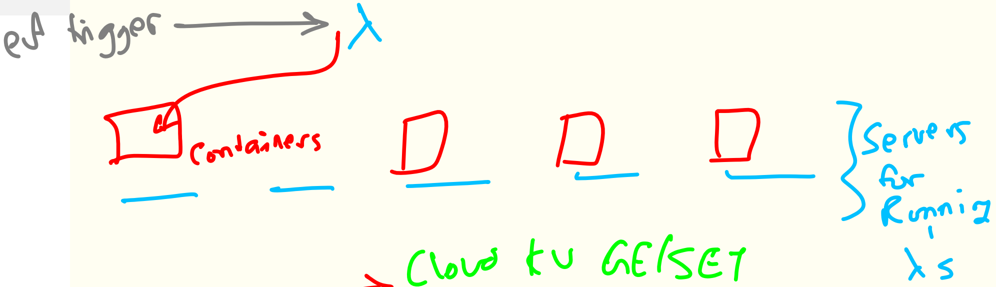
Functions as a Service

λ: functions in FP

- AWS Lambda (∼ 2015)
- FaaS ≡ “Serverless”
- No need to manage VMs explicitly
- Functions run inside sandboxed runtimes

FaaS is a new programming model :

- Functions are “pure functions” and stateless
- Each function invocation is in a new sandboxed environment
- All state is stored in cloud storage services (like S3 buckets)



```
λ' count = 0 → Cloud KV GET/SET
  def handle (fn):
      if (count) ←
   X  ++count;
    return
```

Pricing

- Function invocations are charged
- Pay in proportion to usage
- Application is not charged if not used!
- Resource limits on CPU/memory utilization
- Current pricing is very low: \$ 10^{-7} per request

, NOT allocation

Rare / Sporadic / Bursty events

Function triggers

1. Explicit HTTP requests
2. Changes in cloud storage state (new bucket is added etc)
3. Queuing service (new items added in queue)
4. Publish/Subscribe changes

HTTP Trigger Example

1. Calling a function:

```
curl -X POST
"https://region-project-id.cloudfunctions.net/function-name"
-H "Content-Type:application/json"
--data '{"name":''foo''}'
```

2. Register function: `gcloud functions deploy function-name -runtime nodejs8 -trigger-http`

3. NodeJS function to be called:

```
exports.helloHttp = (req, res) => {
  res.send(Hello ${escapeHtml(req.query.name ||
    req.body.name || 'World')}!);
};
```

= Simple Web Server

- Select a VM
- Configure/update OS
- Setup Apache
- Deploy code

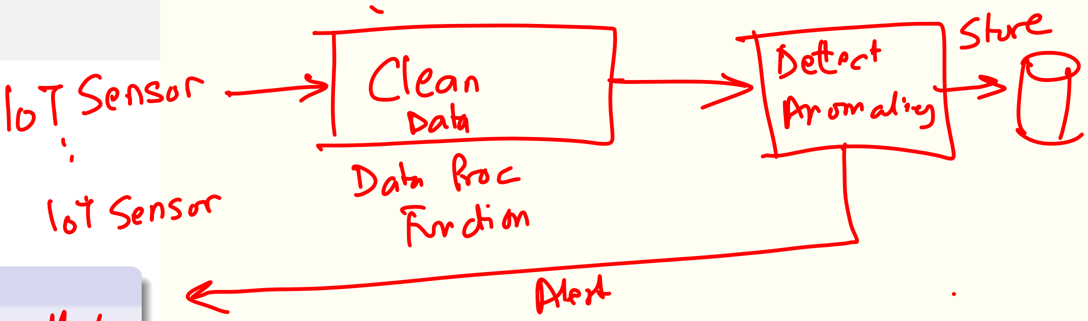
Use-cases

- Web services (static web hosting, blogs, etc.)
- Data processing
- API serving .

low Current costs

New use cases: ← ∞ Scalabili/ Paralleliz

- Highly parallel video processing — 10^4 λ's in parallel
- Linear algebra
- ML training



Implementation

1. On trigger, cloud finds a free server with enough resources
2. Instantiates the language runtime on server
3. Example: If user provides Python code, then launch the Python interpreter, import all dependencies, and run function
4. Once function exits, destroy the execution environment

~100 ms

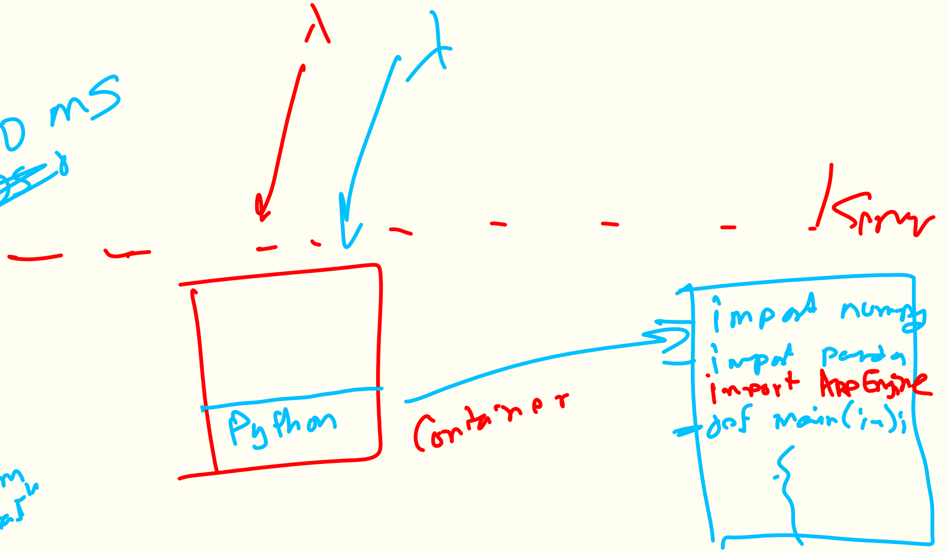
~~~100s~~

Cold-start problem: Function invocation latency can be high

- Frameworks like OpenWhisk, OpenFaaS can be used to setup FaaS on local environments.

keepalive container in memory future invocations

"warm start"



```
import numpy
import panda
import AppEngine
def main(in):
    {
```
~~~

Performance Issues

- Limited lifetimes ($\sim$ 15–30 minutes)
- Storage-bound: excess communication and storage overhead
- Startup-latency: $\sim$ 100 ms . Bad for response times
- No state: Can't use any caching or batching

History

- CGI (Common Gateway Interface)
- Google AppEngine

Pitfalls and Challenges

- Vendor lock-in.
- High costs for applications with steady workloads
- Restricted programming and manageability flexibility
- Heterogenous hardware: using GPUs?