This is a new field, so the terminology is not well defined — the ideas are what is important. We define *cloud computing* as computation done on a network of servers maintained by a cloud service provider

With our model of cloud computing, our goal is effortless elasticity. We can measure this with a few heuristics:

- scalability := (ability to change size)
- availability := (ability to run even with failure)
- fault tolerance := (transparent availability)
- configuration := (ease of setup)
- Quality of Service
- QoS Monitoring Quality

Note: developers often believe the STATE is not part of the QoS metrics but providers disagree.

Cloud computing can be used to the benefit of IOT/Edge Computing, since many home devices now connect to the internet and cannot do heavy computation.

To speed up our computing we use *Fog Computing*.

![](https://paper-attachments.dropbox.com/s_827932EFCCF3E2DE4091C3104E266F6A36BEAB5457E50266A8CDB2D

- localized nodes provide a hybrid between the device and the cloud
- the cloud functions like a cache for computation
- this forms a computation hierarchy

We want to make this *transparent* := to hide the servers from developers entirely, so we run provided functions in the servers, and developers are billed on usage, and it scales to $0.

Models of Cloud Computing

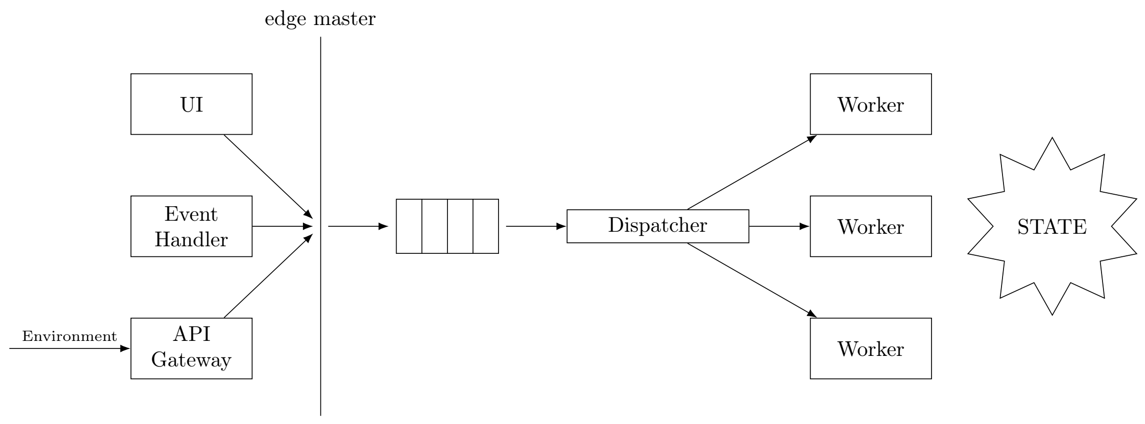
Function at a Service (FaaS)

- A function is the primary unit of computation.
- Functions are executed in response to a trigger (such as an HTTP/HTTPS request).
- Rules:
  - a function must execute relatively quickly
  - a function can have no persistent state
  - functions can, however, refer to storage or database servers to find a state)

For an example of FaaS, consider:

```
function main(params, context) {
  return { payload: 'Hello ' + params.name };
}
// params and context are JSON objects: a marshaling of a data structure
// context is a JSON object that represents metadata like security
```

If we are worried about performance, we would never use FaaS: (un)marshaling is slow. We instead use FaaS for processing speed or functionality.



The steps of event execution are:

1. arrival
2. validation
    (a) authentication
    (b) authorization
    (c) resource limit checking
3. enqueue
4. dispatch
5. allocate a container (a cheap VM)
6. copy function code into the container *
7. execute the function
8. deallocate the container

* our bottleneck is here, since Linux booting is slow relative to all the other operations.

This is called the *cold start problem.*

- This causes our latency to skyrocket.
- We can temper the throughput by adding more workers, but how do we temper latency?
    - pre-load stem cell containers with modules we expect to use
    - do not clear warm containers post use
- We can get to an almost reasonable speed with these techniques!

We call this model the *Action and Trigger Model*

- An event triggers the execution of an action.
- One event can trigger multiple actions. We can use this to implement parallel execution.
- One action can triggers an action. We can use this to implement serialized execution
- this has problems:
    - debugging
        - GDB is too big to use! We have to use their logs!
    - fixing bottlenecks
        - We can't use ps! We have to check usage in the logs post-run!
    - atomicity(?)
        - We are sometimes guaranteed atomicity of function calls
    - change
        - We have difficulty refactoring (breaking functions) and reverting to old versions.
        - The tools are evolving too quickly for us to get used to them!

We have a few other basic models of cloud computing which provide different levels of services

Infrastructure as a Service (IaaS)

- The provider provides supples virtual machines

- A stripped OS called a hypervisor hosts each VM; common ones include Zen and Virtual Box
- The programmer is responsible for the OS, configuration, and applications
- This is 2/3 of enterprise level IT spending. Security concerns keep it from reaching %100.
- There are some major issues, though:
  - We have to anticipate resource usage.
  - It is hard to adjust usage on the fly.

Platform as a Service (PaaS)

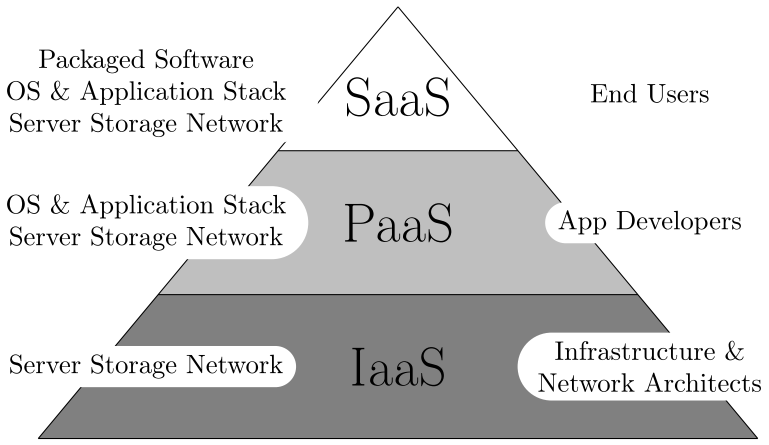
- The provider supplies some of the software stack in addition to VMs
- Less expertise is needed, but provisioning is still pushed on the developer
- Developers still have to autoscale.

Backend as a Service (BaaS)

- the provider supplies a framework for an (often mobile) app on the backend and client
- avoids heavy computation on mobile devices
- scales, but allows little flexibility

Software as a Service (SaaS)

- provider supplies the entire platform and stack
- the developer calls the service's API (which is application specific)
- can be tailored with callbacks in config files



Our models form a pyramid of increasing ease but decreasing flexibility.