# 21. Cloud Computing

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

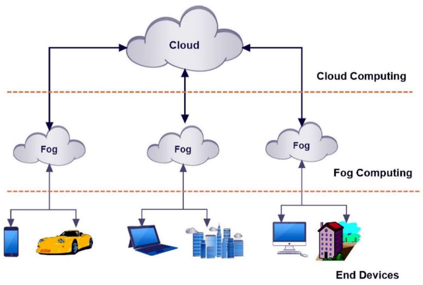
With our model of cloud computing, our goal is effortless elasticity
We can measure this with a few heuristics:

- 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

Cloud computing can be used to the benefit of IOT/Edge Computing

- many home devices now connect to the internet
- devices which are now connected to the internet cannot do heavy computation

To speed up our computing we use Fog Computing



- 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
- developers are billed on usage, and it scales to $0

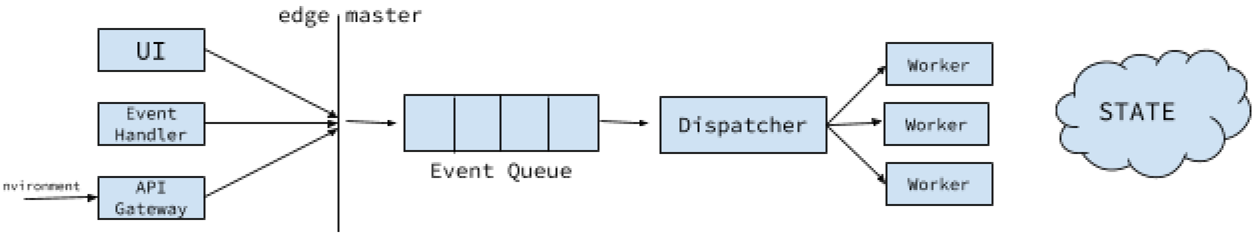
Function at a Service (FaaS)

- 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)
- ex)

```
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 use FaaS for processing speed or functionality



*Note that the workers are virtual machines implemented by Docker (for example)*

- the developers believe the STATE is not part of the QoS metrics — the providers disagree

The steps of event execution :

1. arrival
2. validation
    a. authentification
    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 step (6) 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
    - How do we temper this?
        - 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
    - a model where 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 cause an event that 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 basic models of cloud computing which provide different levels of services

- Infrastructure as a Service (IaaS)

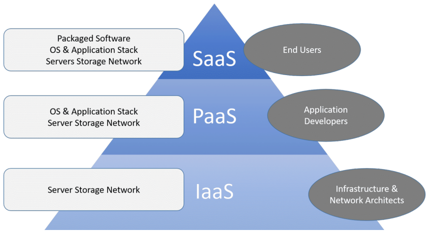
    - the provides supples virtual machines
    - a stripped down OS called a hypervisor hosts each VM
        - common hypervisors include Zen and Virtual Box
    - the programmer is responsible for the OS, configuration, and applications
    - at the enterprise level, this is 2/3 of IT spending
        - it will not reach 100% because of security concerns
    - 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 & stack
    - the developer calls the service’s API (which is application specific)
    - can be tailored with callbacks in config files

Cloud Service Models



Our models form a pyramid of increasing ease but decreasing flexibility