Dynamic formation of the distributed μ clouds

Miloš Simić

— Ph. D. Thesis —

University of Novi Sad
Faculty of Technical Sciences

Table of contents

Research Questions

1. *Can geo-distributed nodes be organized in a similar way to the cloud, but adopted for the different environment, with clear separation of concerns and familiar applications model for users forming micro cloud a model?*
2. *Can these organized nodes, micro clouds, be offered as a service based on the cloud pay as you go model, to the developers and researchers so that they can develop new human-centered applications?*
3. *Can a model be created in such a way that it is formally correct, easy to extend, understand and reason about?*

List of publications

This thesis is the result of research and development, and it is based on the previously presented papers at conferences in journals.

- Journal publications:
  1. **Simić M.**, Prokić I., Dedeić J., G. Sladić and Milosavljević B., "Towards Edge Computing as a Service: Dynamic Formation of the Micro Data-Centers," in IEEE Access, vol. 9, pp. 114468-114484, 2021, doi: 10.1109/ACCESS.2021.3104475
  2. **Simić, M.**; Sladić, G.; Zarić, M.; Markoski, B. Infrastructure as Software in Micro Clouds at the Edge. Sensors 2021, 21, 7001 https://doi.org/10.3390/s21217001

- Conference papers:
  1. **Simić, M.**, Stojkov, M., Sladić, G., Milosavljević, B. CRDTs as replication strategy in large-scale edge distributed system: An overview. In: Zdravković, M., Konjović, Z., Trajanović, M. (Eds.) ICIST 2020 Proceedings Vol.1, pp.46-50, 2020, ISBN 978-86-85525-24-7.
  2. **Simić M.**, Stojkov M., Sladić G., Milosavljević B., Zarić M.: On container usability in large-scale edge distributed system, 9. International Conference on Information Science and Technology (ICIST), Kopaonik: Society for Information Systems and Computer Networks, 10-13 March, 2019, pp. 97-101, ISBN pp.97-101, 2019.
  3. **Simić M.**, Stojkov M., Sladić G., Milosavljević B.: Edge computing system for large-scale distributed sensing systems, 8. International Conference on Information Science and Technology (ICIST), Kopaonik: Society for Information Systems and Computer Networks, 11-14 March, 2018, pp. 36-39, ISBN 978-86-85525-22-3.
  4. **Simić M.**, Sladić G., Milosavljević B.: A Case Study IoT and Blockchain Powered Healthcare , 8. PSU-UNS International Conference on Engineering and Technology - ICET, Novi Sad: University of Novi Sad, Faculty of Technical Sciences , 8-10 June, 2017, pp. 1-4, ISBN 978-86-7892-934-2.

Background

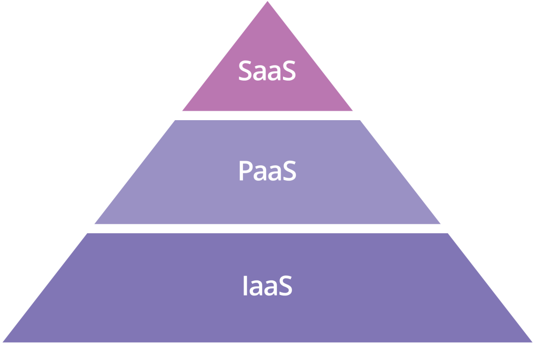
- The entire research in the domain of three main components:
  1. **Distributed systems** and its applications:
     - Cloud computing
     - Big Data
     - Service-oriented architectures
  2. **Infrastructure as software** and its applications:
     - Automation
     - Infrastructure abstraction
  3. **Containers** and their possibilities in software systems
- They are crucial for understanding the concept of **distributed clouds – μ clouds**

Distributed systems

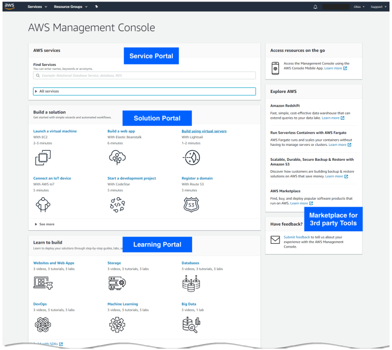
- Multiple entities can communicate, but at the same time, they can perform some operations
- Interesting assumptions:
  - A computing element – either a hardware device or a software process
  - Clients believe they are dealing with a single system – meaning that the nodes need to collaborate
- Three significant characteristics are:
  1. **Concurrency** – multiple activities are executed at the same time on multiple nodes
  2. **Independent failures** – nodes fail independently
  3. **Lack of a global clock** – each node has its notion of time

Cloud computing

Take 1: pyramid of offered services



Take 2: favorite provider



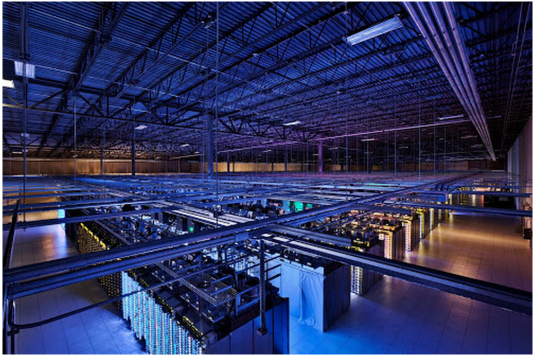
What is the cloud then?

- Vogels et al. describe cloud computing as an **aggregation of computing resources as a utility, and software as a service** — interesting definition
- Clouds provide services, clients use them to avoid **huge** investments – creating and maintaining **big** data centers (**expensive**)
- Clients **only** pay for their usage time – *pay as you go model*
- Centralization of resources – **economy of scale**, lower the administration costs
- **BUT** data **must** to be moved to the cloud – **introduces high latency**

Third time's a Charm — an engineering point of view

There is no cloud just someone else's computer

(Outside Google Data center)



(Inside Google Data center)

And we need to have the way to program these groups of machines

Cloud architecture

- **Cluster** – a set of nodes (resources) that operate as a unit to achieve some goal
- **Region** – a set of clusters, isolated and independent from each other
- Regions are usually composed of a few **availability zones** – defense against the fail
- If one zone fails or goes offline, there are still more of them to serve requests – better availability, scalability, and resilience

- On top, services are built to fully utilize cloud infrastructure

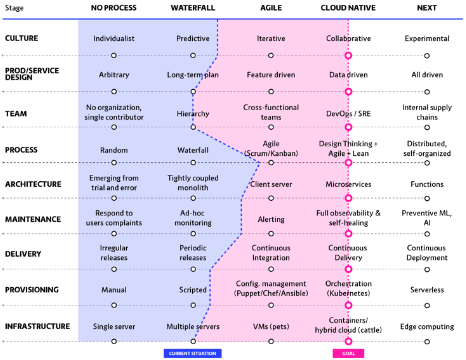
Cloud computing problems

- Centralization **demands** to move all data to the cloud — big issue
  - Boeing 787s per single flight generates **half a terabyte of data**
  - A self-driving car generates **two petabytes of data per single drive**
- Bandwidth is not large enough to support such requirements
- Some applications require real-time processing for proper decision-making
- Such applications might face **serious issues** if a cloud service becomes unavailable
- Some sensitive data **cannot** be moved out of the state border
- When pushed to its limits, centralization brings more harm than good

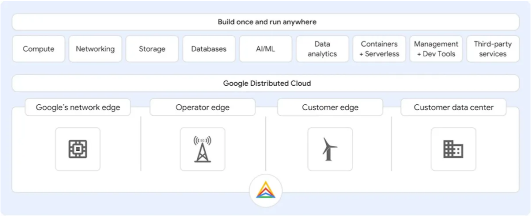
Distributed clouds

- The cloud is usually far away from end devices – **data centers are built on specific locations** (target as many users nearby as possible)
- Sparse deployment will most likely lead to high latency, and bad quality of experience (QoE)
- Latency-sensitive applications **especially** will have a hard time
- Relax the cloud, and move some tasks from the cloud – opening the door for next gen models
- Models where computing and storage resources are in **proximity to data sources**

Next gen



(Cloud Native Maturity Matrix)



(Google distributed cloud)

Edge computing

- A model where computing and storage utilities are **in proximity** to data sources
- It introduced **small-scale servers** – operate **between data sources and the cloud**
- These small-scale servers
  - Have much fewer capabilities compared to the cloud servers
  - Operate in proximity to data sources
  - Maintain good performance to build servers and clusters
- To avoid latency and huge bandwidth, these servers can be dispersed in various locations
- Nearby nodes could be **organized locally – extending resources beyond the single node or group of nodes**

Related work

- Platform models:
  - Kubernetes – an (updated) open-source variant of Google orchestrator Borg
  - Nebula
  - OpenStack
- Nodes organization:
  - Zone-based organization
  - Micro data centers – serve nearby population (theory)
  - Nano data centers – serve single household (using SDN)
  - Drop computing – ad hoc formation
- Processing (Big Data):
  - Data locality – moving computation to the data, instead of vice versa (cloud)
  - Lambda architectures
- Infrastructure definition as code

Proposal

- The cloud architecture is separated into building blocks making the system easier to understand, maintain and operate – clusters, regions, availability zones
- Rely on a similar proven strategy – adapt the existing model for different use-cases

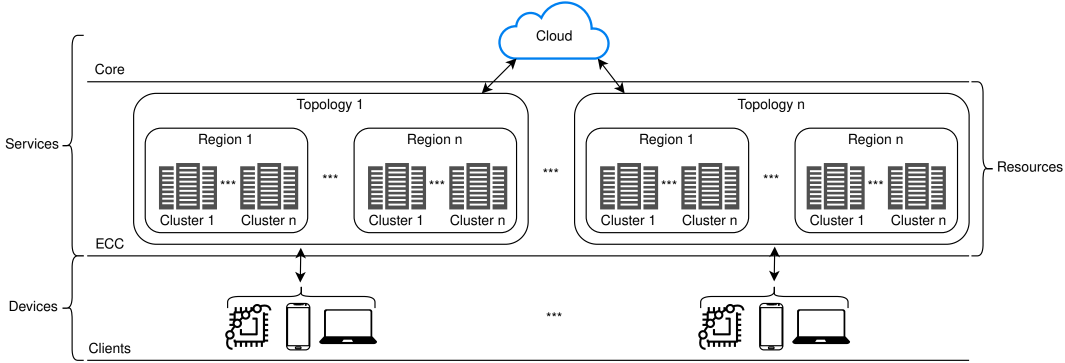
- Let's observe **micro clouds** as *geo-distributed systems with dispersed users*
- *Geo-distribution* means **in proximity to some large populations**, where micro clouds serve user requests locally **first**
- Forming micro clouds (pools of resources) depends on local population **usage and needs** – adaptive
- Reduce traditional data centers fixed costs with **agility** – dynamically grow and shrink resources as needed

Organization of the nodes

- A group of nodes virtually and/or geographically separated, working together to provide the same service to clients, forms a **cluster**
- A concept used to describe a set of clusters (that could be) scattered over an arbitrary geographic region, is a **region**
- Highest logical concept that is composed of a minimum of one region and could span over multiple regions is a **topology**
- **To lower the latency, vast distances between clusters should be strongly avoided in normal circumstances!**

Edge centric computing	**Cloud computing**
Topology (logical)	Cloud provider (logical)
Region (logical)	Region (physical)
Cluster (physical)	Zone (physical)

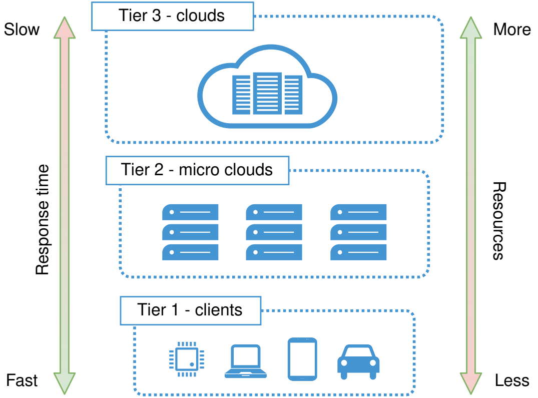
Separation of concerns



(Edge-centric computing as a service architecture with separation of concerns.)

Micro clouds

- **Ephemeral cloud-like structures** serving local requests first, before reaching to the traditional cloud
- Size and existence are defined by the **local population's needs**
- Clients have the **illusion** to communicate with the clouds
- Designed for **failure** using automated tools
- **Collaborate** with traditional clouds – biologically inspired computing



Physical capabilities

- Capabilities of ARM-based devices have good performance for building servers and clusters, considering their performance per Watt relation
- These servers can be spread in base stations, coffee shops, or over geographic regions to avoid latency, and huge bandwidth
- They can serve as firewalls and pre-processing tier for the cloud (smart (big) data)
- Users get a unique ability to dynamically and selectively control the information sent to the cloud

Who can join the system?

To be a part of the system, a node **must** satisfy four simple rules:

1. Run an operating system with a file system
2. Be able to run some application isolation tool (e.g. container or unikernel engine)
3. Have available resources for utilization
4. Have internet connection

- Simple yet powerful rules are here to help – increased demand for resources that the currently available infrastructure **cannot** support
- **Inclusion of volunteer nodes** into the system can be allowed to depreciate load for an indefinite period

But what is a node?

- Nodes can be **heterogeneous** by their nature
- Formally, every node in the system could be described as a *tuple*

$$s_i = (L, R, A, I)$$

- L is a set of ordered *key-value* pairs representing node *labels* or **server-specific features** (e.g., os:linux, arch:arm, isolation:containers, etc.)
- R is representing node resources (kind, total, free, used, etc.)
- A is representing running applications (labels, resources, configrations, informations, etc.)
- I represents a set of general node information like (name, location, IP address, id, cluster id, region id, topology id, etc.)

Formation of micro clouds and the protocols

- Nodes are organized into micro clouds dynamically, **by abstracting infrastructure to the level of software** — infrastructure as software
- The proposed system relies on **four** protocols:

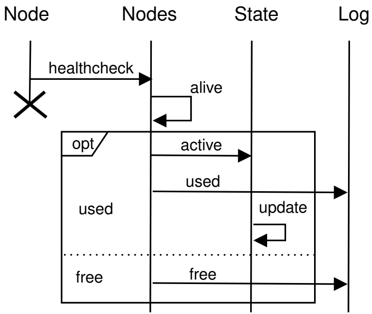
1. **healthcheck** protocol informs the system about the state of every node — background operation
2. **cluster formation** protocol forms new clusters — mutate operation (atomic, immutable, and idempotent)
3. **Idempotency check** protocol prevent mutate operation **if** infrastructure already exists
4. **list detail** protocol shows the current state of the system to the user — list operation

Healthcheck protocol

- The **order** in which messages arrive is **not** important
- The important part is that every message **eventually** comes into the system
- All nodes are **equal**, part of some cluster or not
- Formally adding node to the system **if not present**:

$$S_{new} = S_{old} \cup \bigcup_{i=1}^{n} \{s_i\}$$

- Othervise update node state



Cluster formation and Idempotency check protocols

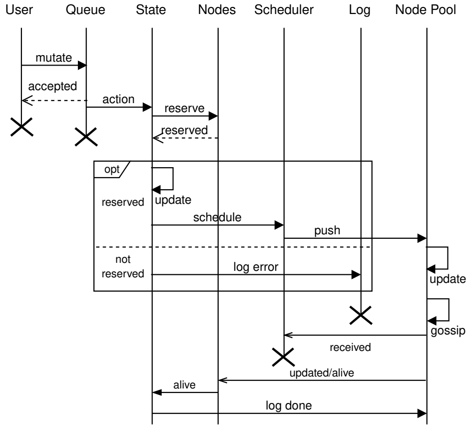
- Pick desired nodes, rules:
  - i_{th} server's labels set $s_i[L]$ and the query selector Q are same size

$$|s_i[L]| = |Q|$$

  - every key-value pair from query set Q is present in the i_{th} server's labels set $s_i[L]$

$$P(Q, s_i) = \Big(\forall (k, v) \in Q \, \exists (k_j, v_j) \in s_i[L] \text{ such that } k = k_j \wedge v \leqslant v_j\Big)$$

- Check idempotency
- Reserve nodes for some time T_r
- Nodes start **membership protocol**



Cluster formation and Idempotency check protocols

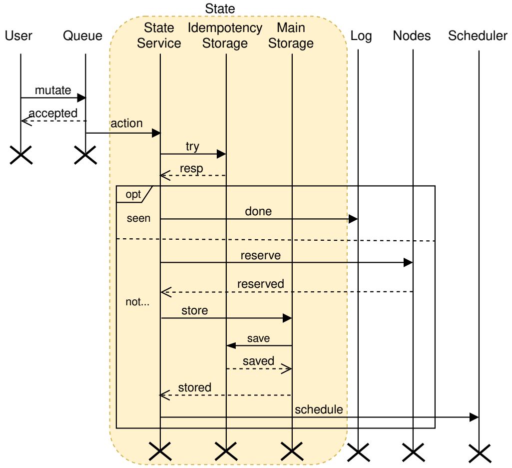
- Pick desired nodes, rules:
  - i_{th} server's labels set $s_i[L]$ and the query selector Q are same size

$$|s_i[L]| = |Q|$$

  - every key-value pair from query set Q is present in the i_{th} server's labels set $s_i[L]$

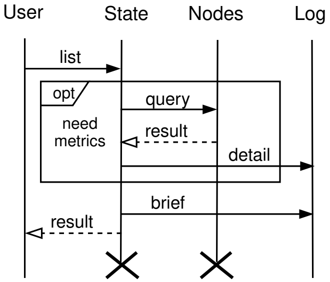
$$P(Q, s_i) = \Big(\forall (k,v) \in Q\ \exists (k_j, v_j) \in s_i[L] \text{ such that } k = k_j \land v \leqslant v_j\Big)$$

- Check idempotency
- Reserve nodes for some time T_r
- Nodes start **membership protocol**



List detail protocol

- Information retrieval protocol — dashboard
- Two options:
  1. **global view** of the system – basic informations
  2. **specific clusters** details – in-depth details for specified clusters
- When picking nodes, both rules **must** be satisfied (similar to cluster formation protocol)

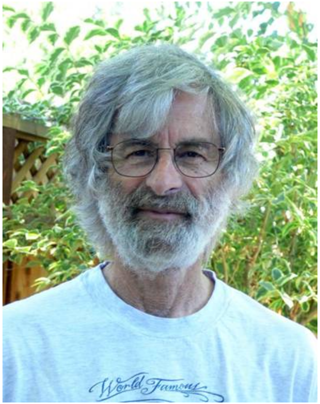


$$R = \{s_i \mid |s_i[L]| = |Q| \wedge P(Q, s_i), i \in \{1, \dots, n\}\}$$

How to prove this?

If there is a bug in a *distributed algorithm*,
no matter how improbable it may seem,
it's not a question of **whether** it will appear,
it's a question of **when** it will appear.

(Leslie Lamport, Heidelberg Laureate Forum 2021,
https://www.youtube.com/watch?v=KVs3YFKqclU)



(Leslie Lamport, Turing award amongst others)

Formally specifying our protocols

- Formal analysis
- Multiparty asynchronous session types
- A true taste of computer science

We were searching for:

- a formalism that is proven correct
- and expressive enough
- but also easy to follow

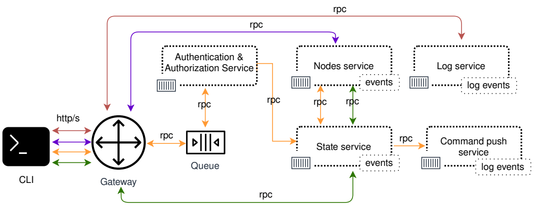
(twitter.com/heidiann360/status/1332711011451867139)

- Joint work with Ivan Prokić and Jovana Dedeić from Chair of Matematics FTN

Real-life implementations — enter constellations

We see two possible scenarios for implementation:

- **a stand-alone** implementation — using open source tools
- **integration** within existing tools, as a node organizer and register (e.g. Kubernetes node organizator)
- **Do not** lock users in specific provider ecosystem



(Proof of concept implementation — constellations project (c12s))

Remote formation is tricky

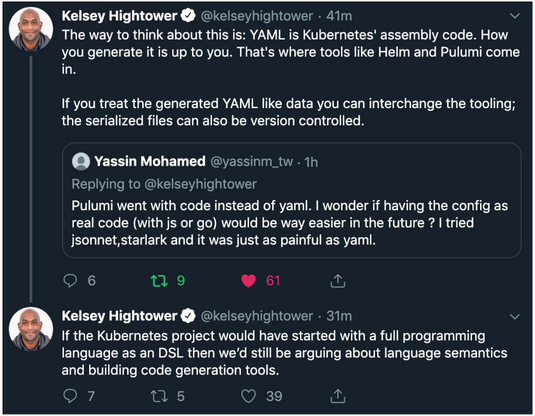
- Infrastructure deployment will **not happen overnight** – it might take years
- It might not be started at all until the whole process is *trivial* – **key is to simplify remote micro clouds management**
- The naive approach, do it manually – super tedious and time-consuming
- **The infrastructure needs to be constantly deployed and maintained** – automation is the key
- Prevent configuration drift – the systems become different over time

Then how?

- Abstracting infrastructure to the level of software – **infrastructure as software** (already available tools, principles, and techniques)
- Micro clouds model is **movable** in between edge and cloud – do we want more cloud-like or edge-like system
- Such **elasticity** requires infrastructure abstraction to the software level – **infrastructure programming**
- This allow micro clouds infrastructure to be managed similarly as the software is
- Infrastructure definition is versioned, automated, and applied repeatedly and consistently every time – **minimize configuration drift**

Infrastructure programming artifacts

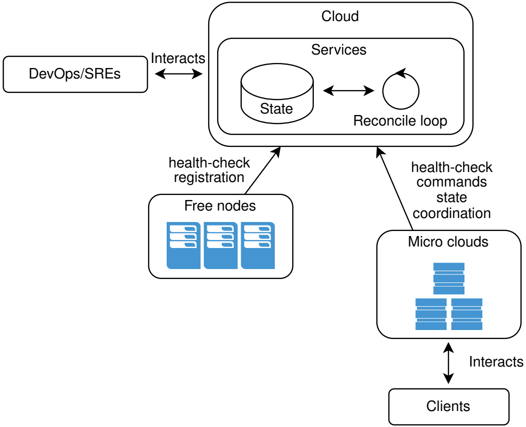
- Specify **desired state** using *YAML* or some other format – descriptively
- Submit **desired state** to the system, and **let the system deal with the rest**
- **Immutable infrastructure deployment** allows for rolling update strategy – updates large environments, a few nodes at the time
- Rely on containers for everything, even OS (LinuxKit)
- Sorry **no** DSLs here



(Source: Twitter)

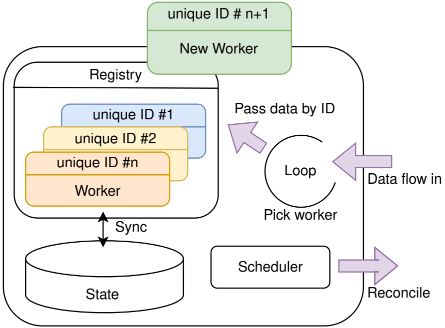
The reconciler pattern

- System is dealing with changes using the **reconciler pattern**
- Track resources using two simple states:
  1. **expected state** – desired state
  2. **current state** – actual state
- **Reconciliation loop** ensures that the desired state is maintained
- Every node **must** provide current state, for comparison – **healthcheck** protocol
- Extension is simple



The reconciler pattern

- System is dealing with changes using the **reconciler pattern**
- Track resources using two simple states:
  1. **expected state** – desired state
  2. **current state** – actual state
- **Reconciliation loop** ensures that the desired state is maintained
- Every node **must** provide current state, for comparison – **healthcheck** protocol
- Extension is simple



What exists?

- The existing orchestrator engines operate on a cluster level — "treating **servers** as cattle, not pets" (i.e. numerous **servers** built using automated tools designed for failure, where no **server** is irreplaceable) — cluster could span over multiple availability zones
- Kubernetes allows multi-cluster deployments, handling these clusters as disposable — "treating **clusters** as cattle, not pets" (i.e., numerous **clusters** built using automated tools designed for failure, where no **clusters** are irreplaceable)

And what's new – challenging the state of the art?

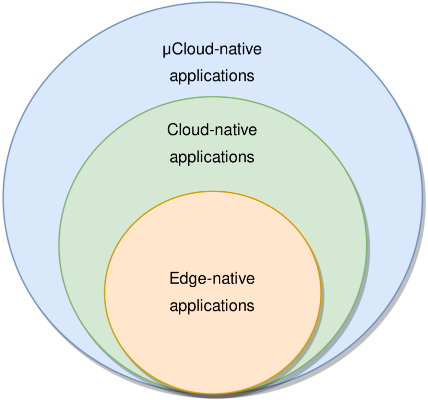
- This research goes one step further, allowing the creation of numerous micro clouds as disposable, using automated tools where no micro cloud is irreplaceable — "treating **micro clouds** as cattle, not pets" (i.e. numerous **micro clouds** built using automated tools designed for failure, where no **micro cloud** is irreplaceable)
- More dimensions for operation and optimization of infrastructure and services
- Allowing mIaaS, mCaaS, mPaaS, mSaaS models closer to the data data sources

Limitations

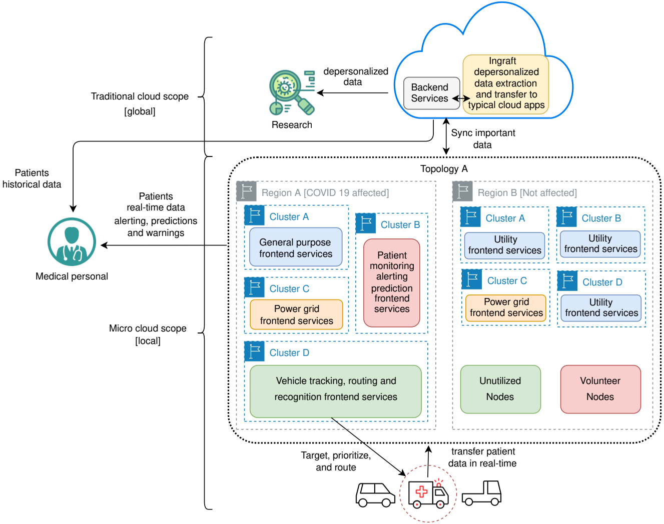
- Specialized models are developed and optimized for a specific use case to take the maximum out of hardware and software
- They might outperform the proposed model in terms of speed
- The proposed model allows organization and reorganization of resources in a similar way the cloud does, allowing users to develop applications without some specialized infrastructure for different applications — more development freedom
- Users will be able to create new interesting human-centered applications in the future, utilizing both cloud and edge
- The proposed model is more oriented towards developing a broader specter of applications without the need for a specialized hardware or software

Applications

- All existing applications in the cloud can be upgraded to use the new model
- Big data applications **AND/OR** smart-sized, "data-centric" solutions to big issues – Andrew Ng
- Real-time processing for proper decision making
- Eventually – OS capable of running infrastructure with **less to none** human intervention



Use cases


Traditional cloud scope
[global]
Research
depersonalized data
Backend Services
Ingraft depersonalized data extraction and transfer to typical cloud apps
Sync important data
Patients historical data
Patients real-time data alerting, predictions and warnings
Medical personal
Micro cloud scope
[local]
Topology A
Region A [COVID 19 affected]
Cluster A
General purpose frontend services
Cluster B
Patient monitoring alerting prediction frontend services
Cluster C
Power grid frontend services
Cluster D
Vehicle tracking, routing and recognition frontend services
Region B [Not affected]
Cluster A
Utility frontend services
Cluster B
Utility frontend services
Cluster C
Power grid frontend services
Cluster D
Utility frontend services
Unutilized Nodes
Volunteer Nodes
Target, prioritize, and route
transfer patient data in real-time

Concluding remarks

The dynamic organization of geo-distributed nodes into micro clouds, forming distributed clouds

Our model:

- Nodes are organized into micro clouds dynamically – inspired by the cloud architecture
- Creation of numerous micro clouds designed for failure using automated tools where no micro cloud is irreplaceable – "treating **micro clouds** as cattle, not pets."
- Serving local requests first, and from optimal location
- Abstracted infrastructure to the level of software – infrastructure as software
- Moving computation to the node where that data resides, instead of vice versa

"Containers are great, let's run them everywhere." – Brian Dorsey, Google Cloud Platform.

Future work

- We barely scratched the surface, and there is still a lot of work to do
  - Remote configurations and secrets management – work in progress
  - Hierarchical idempotence – work in progress
  - Namespaces and virtual clusters – work in progresss
  - Autoscaling – scale everywhere
  - File system/storage layer – write once store everywhere
  - Security aspects – Security as code
  - Applications framework
  - Integration with existing infrastructure
  - Lambda++ architecture
  - Scheduling
  - Chaos engineering
  - Monitoring
  - Replication
  - etc.
- Not lacking opportunities

Thank you for your attention!