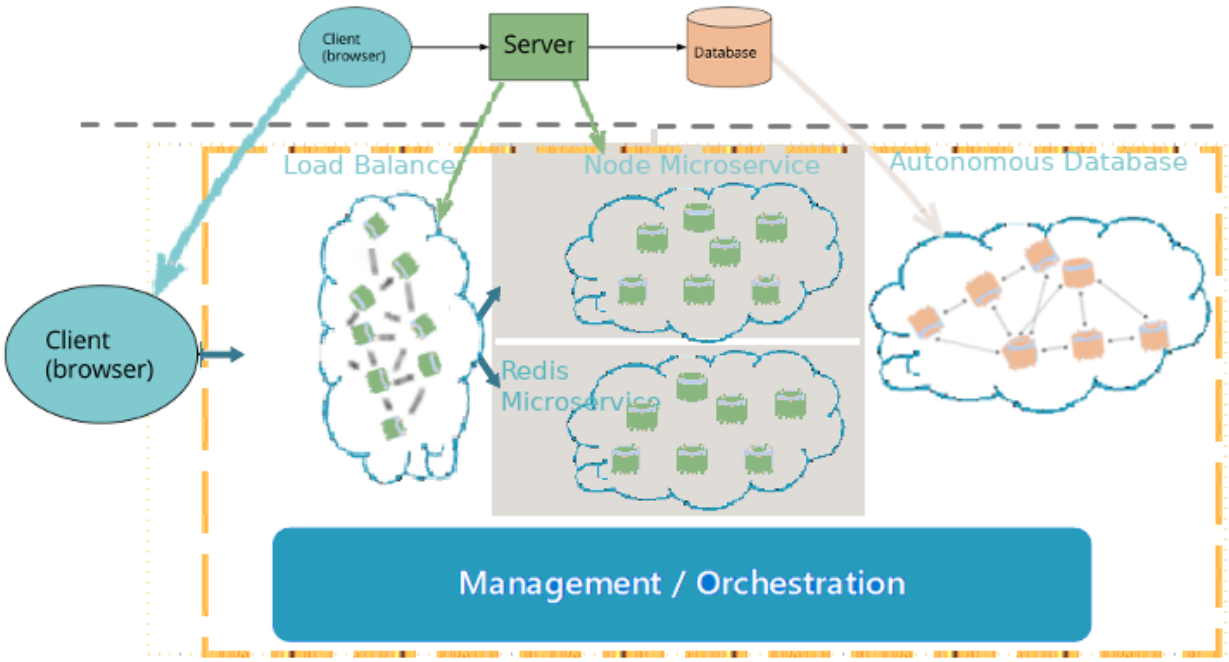
Architecture!

Table of contents

Architecture

We will be trying to create a swarm implementation that will allow communication between all of the members/nodes.

1. Deploy microservices to a Swarm cluster



microservices architecture

- a microservice performs a simple task
  - communicates with clients or
  - other microservices
    - communication mechanisms such as REST API requests or Websocket etc

- Microservices can include any programming language you like
  - and with the orchestration tools they are easy to deploy and maintain

```
This architecture uses NodeJS and Redis microservices deployed
as Docker containers
```

1.1. Scalability

You can scale your application by updating the number of replica nodes in the swarm cluster

```
...
    deploy:
      replicas: 15
      placement:
        max_replicas_per_node: 1
        constraints:
          - node.labels.region==regio1
...
```

Segmentation

Dynamic infrastructure: services can scale up and down without waiting for each other.

1.2. Availability

No single point of failure.

1.3. Security

1.4. Configuration

2. Some thoughts regarding Microservice-based application

Distributed Service Placement for Microservice-based Applications

Nowadays whith applications becoming more and more powerfull, users more demanding and IOT devices taking swarming our lives, the need for maximizing ***location awarenes*** and minimizing delay has entered the field.

Thus we cannot relliably trust a signle central data point and have to mingrate to versatile and distributed systems moving the brain of our application ever closer to the user (edge).

Multi-access Edge (MEC) Computing was born.

Location-aware technology is any technology that is able to detect its current location and then analyze this data to control event and information flow. Wikipedia

MEC can be defined as cloud services running at the edge of a network and performing specific tasks — in real- or near-real-time wikipedia

To adapt to the above we can shift towards:

- small and scalable data-centers placed closed to the cloud edge
- minimalization of backbone data transmission

2.1. this tools will help

Container technologies (Docker), and **orchestration/maintenance tools** (Kubernetes, DockerSwarm etc), are becoming the mainstream solution for packaging, deploying, maintaining, and healing applications.

Each microservice decoupled from the application can be packaged as a Docker image and each microservice instance is a Docker container.

Kubernetes for example is one of the best examples for creating cloud-native applications and leveraging the benefits of a distributed system.

2.2. and here is the problem

Problem:

When all of the services are placed on one edge site, network congestion is inevitable.

Solution:

With one service deployed on more than one edge site, requests from different end users at different locations can be balanced, so as to ensure the high availability of service and the robustness of the provision platform.

But, most tools used have some hard limits concerning this.

- It is often treated as a single abstract service with given input and output data size.
- Time series or composition property of services are not fully taken into consideration.
- Due to the heterogeneity of edge sites, such as different CPU cycle frequency and memory footprint, varying background load, transient network interrupts and so on, the service provision platform might face greatly slowdowns or even runtime crash.
- the default assignment, deployment, and management of containers does not fully take the heterogeneity in both physical and virtualized nodes into consideration.
- Besides, the healing capability of Kubernetesis principally monitoring the status of containers, pods, and nodes and timely restarting the failures, which is not enough for high availability.
- in some cases when the pod failure happens, the outage time of the corresponding service could be dozens of seconds.
    - When node failure happens, the outage time could be dozens of minutes

2.3. Suggestion

```
Therefore, we have to create a better maybe even using different tools to better
manage the distribution of our system data and eliminate the problems mentioned above.
```

```
This solution will most likely have to be able to utilize multiple edge
applications/host at once to balance out the load on one single node.
```

However,

- Such kind of service requires redundancy and the amount of hosts needed is difficult to agree upon since the network constantly changes.
- Therefore our best bet is to seperate the different cases and try our best to analyze the data and plan our service.

Another problem , currenty not tackled by any major providers are unique attributes of each location, like traditional or human behavioural patterns (for example parking in cetral Rome or a Greek island!).

Gathering data from different requests and places we have to design an algorithm to better utilize the location-specific information.

Our target would be to slowly adapt to all locations and various unique details of each of them.

- better services for the end users always keeping in mind to best distribute our system

- creation of a general microservice based chain-application that works as one, large, redundant, distributed system