Project:

Optimizing Hardware usage in multi-cloud environment

Teacher: Henry Muccini

Project Partner: Daniele Spinosi, Micron Technology

Deliverable Template

Date	**22/02/2021**
Team ID	**Script L in BUS**

Team Members		
Name and Surname	**Matriculation number**	**E-mail address**
Fabio Gentile	**274716**	**fabio.gentile@student.univaq.it**
Di Egidio Laura	**273860**	**laura.diegidio@student.univaq.it**
Stefano Di Marco	**274981**	**stefano.dimarco4@student.univaq.it**

Executive summary

For this project, we thought about a few different architectures, but in the end we decided on the one that let us spend less Capital.
We decided for this solution: ***Disaster Recovery as a service.***

With this Architecture we invest **396.000 USD** if there is no Datacenter failure.

In the case of a failure we are going to spend: assuming that we can recovery from the **crash of 1 server** in at most 48 hours, 1,856 USD (Pay as you go price per hour) * 48 = **89,1 USD**
If an entire datacenter crashes we assume we can recovery from the crash in at most 1 month, so we are spending at most: 1354,88 * 5 = **6.774,44 USD**

What about the Return on investments (**ROI**)?

Since ROI = Ro/Ci namely Ro = Operating income and Ci = Capital invested, and we are reducing the number of servers without affecting the revenues, our ROI will be greater.

ROI before:
We are only interested in the data we have, so we consider this time-line: Year -3, Year -2,Year -1, Year 0 , Year 1, Year 2. In the years -3,-2,-1 we use the old architecture, in the next 3 years we use our new architecture.

- In year -3 : investment of 10 Servers for 6 Datacenters (3 factories) = 720.000 USD
- In year -2 : 0
- In year -1 : Maintenance costs: 72.000 USD

ROI in year -1:
RO = Revenues - Costs (Including 72.000 USD)
Ci = Total assets (Including Servers 720.000)
ROI = RO / Ci = (RO - 72.000) / (Ci + 720.000)

ROI in year 2:
RO = Revenues - Costs(Including 36.000 USD)
Ci = Total assets(Including Servers 360.000)
ROI = RO / Ci = (RO - 36.000) / (Ci + 360.000)

We assume that RO doesn’t vary if we change architecture and since we decreased costs for maintenance, the numerator will be bigger with respect to the previous one (Year -1). Same happens for the denominator, we have less long term investments, so the denominator will be smaller, this will result in an increase in ROI. An increase in ROI could lead to an increase in profitability.

In Detail:

Before our Strategy:

10 servers * 2 Datacenters * 3 Factories = 720.000 USD + 72.000 maintenance every 3 years = 792.000 USD

After our strategy:

5 Server needed * 2 Datacenters * 3 Factories = 12.000 * 5 * 2 * 3 = 360.000 + 36.000 Maintenance every 3 years = 396.000 USD

So we are going to save the 50% of the costs 792.000 USD - 396.000 USD = 396.000 USD

We have extra costs in case of failure of servers/Datacenters. A Datacenter should crash 58 times in 3 years to reach the costs of the previous architecture.

Calculations: 396.000 / 6774,44 = 58,44

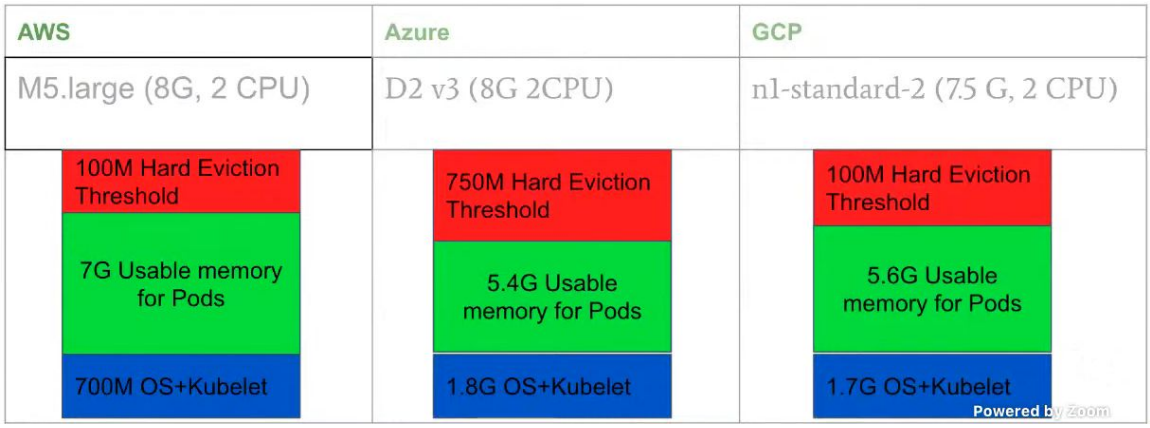
Table of Contents

Challenges/Risk Analysis

Risk	Date the risk is identified	Date the risk is resolved	Explanation on how the risk has been managed
Availability	18/11/2020	10/01/2021	Using cloud services for fault tolerance. We forward 100% (total is 120% 20% will be ran locally) of T2 Services to AWS Public Cloud in a Pay As You Use mode. This is in case of a crash of an entire Datacenter. 20% of T2 will run locally on the adjacent Datacenter.
Asynchronous service communication	22/11/2020	03/12/2020	We decided on Kafka for communication between services.
Knowledge on specific domain	24/11/2020	Never ends	Study the new technologies to use
Build,deployment and orchestration of services	19/11/2020	11/01/2021	Choosing right tools and technologies
Load Balancer work flow	03/01/2021	15/01/2021	We have chosen to balance work flow with Server Iron solutions.
CPU usage 100%	22/11/2020	10/01/2021	As written in the specification we assume that a server can work without problem at 100%. So in a normal situation we have 5 server in each DataCenter that works at 100%
Microservices Architecture switch	27/12/2020	On going	The possibility that to switch from SOA and microservices architecture we can have data loss.

Spec Refinement

Our goal is to swap from a SOA/Mini-services architecture to a full microservices system, the 2 datacenters will run a private cloud Open-Shift. Microservices are advanced with docker, so the microservices will be containerized and will be deployed on the private cloud and will be orchestrated with Kubernetes which is integrated in our platform Open-Shift. Open-Shift is a Platform as a Service. With an open shift pipeline we distribute our service work flow between the on-premise and public cloud (strictly for few non critical services). We chose AWS cloud (Public side) because Kubernetes is easily implementable and as we can see in the following picture, it is the minimum resource consuming one for what regards OS + Kubelet, allowing us to gain more Memory for our pods.



Source*: “All Day DevOps conference ” 12 nov 2020;*Title*: Ruling Kubernetes on the Cloud* - Author*: Huseyin BABAL, Topics: “Architecture and Microservices, DevOps Pipeline, Kubernetes”*

Each factory has 2 datacenters each one having 5 servers inside. So the services are scaled as required by kubernetes and resource consumptions are optimized. T1 services just run on both the datacenters (fault tolerance decision) so if 1 datacenter goes down, the remaining one will run the T1 services entirely.

If a datacenter fails so we remain with 1 datacenter, T1 services will take 80% of cpu usage because it needed 40% when running on 2 datacenters.

So what happens to T2 Services?

The data center that has remained online takes all the T1 for an 80% and the 20% of the remaining computational resources are used for T2; for the remaining 100% needed for T2 we use Pay as you use AWS Public Cloud services.

The services will communicate to each other using a publisher subscriber system, more precisely we will use Kafka.

We use kafka as an asynchronous communication system in a microservices environment, it provides fast communication between the Services and reduces the risk of losing data. A good advantage that brought us choosing Kafka is that it handles the messages with a very low latency in the order of milliseconds that could be very important in a production.
Kubernetes will also be auto-scaling the services, deploying more images of that container and so increasing its availability.

We assume that only allowed staff members can access the buildings where the Datacenters are located. The datacenter must be in a safe building with a certain constant temperature monitored via sensors.
For Blackouts and force majeure events we assume we have Rolls Royce engines to provide enough power to keep up the datancenters for an entire working day.
For WAN failures, we assume that we have a contract with GTT Communications Inc. with a Service level agreement of 99,999% of availability, frame or package loss: ≤0,1%, Frame jitter: ≤2ms.

To switch to the new architecture we assume that the transition time is at most 1 week.

Strategy Architecture studied:

The team have study three architectures strategy that describe as follow:

1. “Raid” Strategy:

We share the fault tolerance to all datacenters. Strategy based on NAS fault tolerance.

Server needed:
We need 6 servers that run at 83% in each datacenter.

How do we guarantee the availability of T1 services in case of failure?

We guarantee it by moving these services to the adjacent datacenter that will take care of these services and share the T2 with the other Datacenters. All datacenters will bring their percentage of cpu work to 100%.

Should we use hardware in the cloud?

No. We don't have to buy or rent any other hardware.

How much does this strategy cost?

6 server needed on each datacenter
6 datacenter

36 Servers x 12.000 USD = 432.000 USD
\+ 10% for maintenance (43.200 USD)
Total = 475.200 USD

Advantage:

High Fault tolerance
Futurability, as data centers increase, the percentage of usable cpu increases
No extra costs

Disadvantage:

Cpu not used at 100%

2. “Rent Server for T2 Services” Strategy:

We only guarantee fault tolerance for T1 services using the adjacent DC and rent cloud servers for T2 services (only the remaining 20%). T1 services make up 40% of the current load. We therefore guarantee 80% of the current load in the two adjacent datacenters and the remaining 20% is rented in the cloud.

Server needed:
We need 4 servers that run at 100% in each datacenter

How do we guarantee the availability of T1 services in case of failure?

We guarantee it by moving these services to the adjacent datacenter that will take care of these services and share the T2 with the Cloud server in Pay-As-You-Use mode.

Should we use hardware in the cloud?

Yes, we must use cloud hardware:
6 leased servers to calculate the remaining 20% (in each DC) of the t2 services.
In case of error of an entire DC we have to use 4 servers in pay-as-you-use mode to compute 80% of the remaining T2 services.

How much does this strategy cost?

4 server needed on each datacenter
6 datacenter

6 server in Cloud

24 Servers * 12.000 USD = 288.000 USD
\+ 10% for maintenance (28.800 USD)
\+ 6 server in cloud x 30.526,2 USD = 183.157,2 USD

Total = 499.957,2 USD
+ any failure that occurred within 3 years (Calculated in the ‘Strategies Analysis’)

Advantage:
Cpu used at 100%
Good Fault tolerance

Disadvantage:
Fault tolerance for a fee
More expensive

3. “Disaster recovery as a service” Strategy: STRATEGY CHOSEN

We use 5 servers up to 100% and we use cloud servers only in case of error.
In case of failure of one DC, we guarantee fault tolerance for T1 services using the adjacent DC and T2 services are sent to the cloud.

Server needed:
We need 5 servers that run at 100% in each datacenter.

How do we guarantee the availability of T1 services in case of failure?

We guarantee it by moving these services to the adjacent datacenter that will take care of these services and share the T2 with the Cloud server in Pay-As-You-Use mode.

Should we use hardware in the cloud?

Yes, we must use cloud hardware:
In case of error of an entire DC we have to use 5 servers in pay-as-you-use mode to compute 100% of the remaining T2 services.

How much does this strategy cost?

5 server needed on each datacenter
6 datacenter

30 Servers x 12.000 USD = 360.000 USD
\+ 10% for maintenance (36.000 USD)

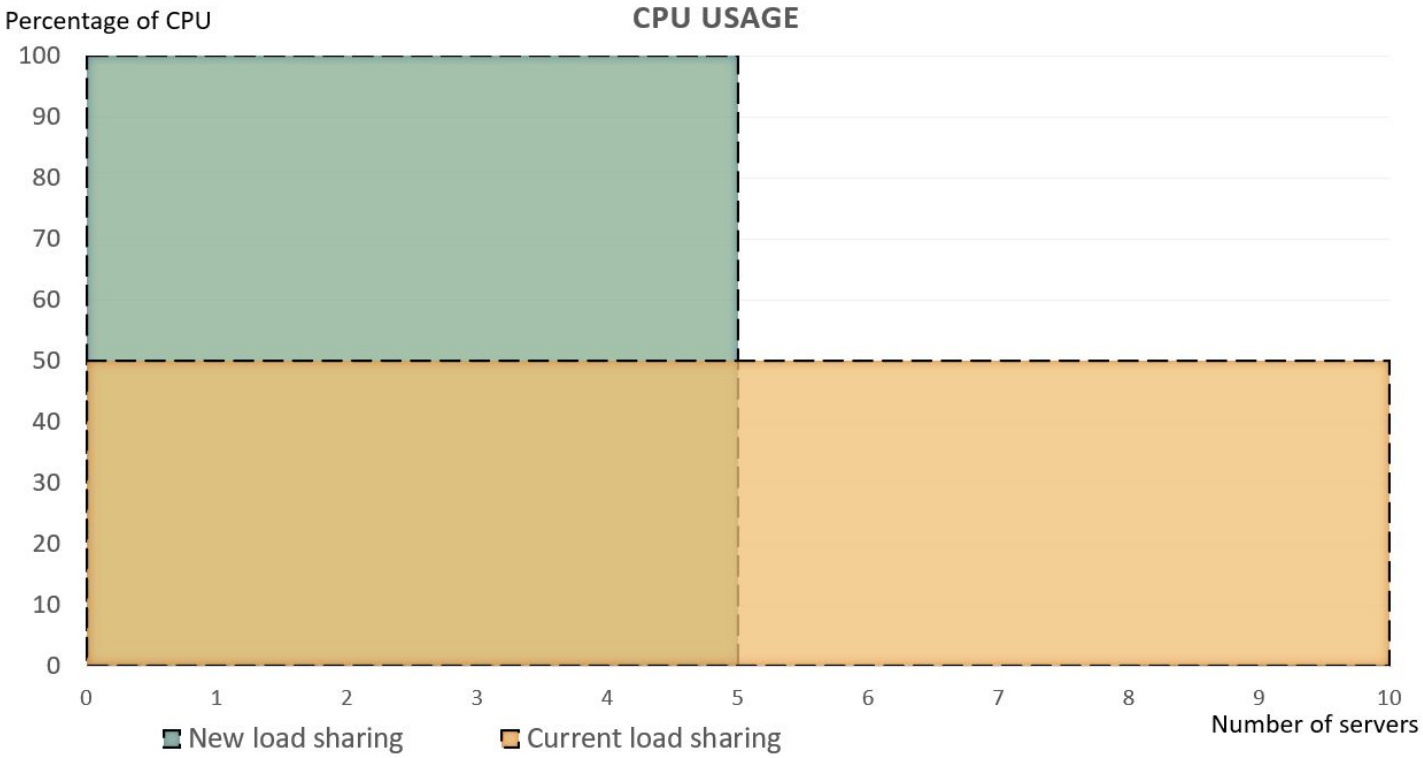
Total = 396.000 USD
\+ any failure that occurred within 3 years (Calculated in the ‘Strategies Analysis’)

Advantage:

- Cpu used at 100%
- Good Fault tolerance
- Reduced costs

Disadvantage:

- Fault tolerance for a fee



Here we represent how to change the CPU usage from current load sharing to the new load sharing.

Strategies analysis

To choose a strategy in particular we must focus on the probability of error and the frequency with which they occur.

We assume that any breakdown of a server and its replacement does not generate additional costs because it is guaranteed for the entire 3-year life cycle by the manufacturer when the supply contract is signed.

For strategies 2 and 3 we talked about a paid fault tolerance, but this naturally depends on the size of the fault and the time required to restore it.

For example, in both strategies, if we have a failure on a single server and 48 working hours are used to repair it, the cost derived from this failure is 89,088USD for the rent in Pay-Use mode. (labor costs and any hardware changes are excluded from the calculation) We do not exclude that on average the recovery of a server takes less time (3.8 hours) and this would mean lower recovery cost.

In Strategy 2 the crash of an entire datacenter means renting 4 servers in Pay-Use mode. Assuming the failure is repaired in at most one month, the cost of using the servers in the cloud is 4 x 1.354,88 USD = 5.419,52 USD (labor costs and any hardware changes are excluded from the calculation)

In Strategy 3 the crash of an entire datacenter means renting 5 servers in Pay-as-you-Use mode. Assuming the failure is repaired in at most one month, the cost of using the servers in the cloud is 5 x 1.354,88 USD = 6.774,44 USD (labor costs and any hardware changes are excluded from the calculation).

In the very unlikely case of 11 crashes that last at most one month of datacenters in 3 years, then the RAID will be better than “Disaster recovery as a service” from a cost point of view.
475.200 - 396.000 = 79.200 => 79.200 / 6.774,44 = 11 Datacenter Crash in 3 years.

Design Decisions

Concern (Identifier: Description)		Con#1: what kind of architecture should we use to manage services?
Ranking criteria (Identifier: Name)		Cr#1: Resiliency Cr#2: Performance Cr#3: Scalability
Options	***Identifier: Name***	Con#1-Opt#1 : Microservice Architecture
	Description	The microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API.
	Status	This option is **decided**
	Relationship(s)	This decision is related to Con#4-Opt#1, Con#4-Opt#2, Con#4-Opt#3
	Evaluation	Cr#1: With microservices, applications can fully handle service failures by isolating functionality without blocking the application. Cr#2: The microservices model allows provisioning only of the parts of the software system that need to be scaled, in a dynamic and smart way. Cr#3: Microservices allow us to scale each service independently to meet the demand for the functionality that the application supports. This allows, for example, to accurately measure the costs of a feature.
	Rationale of decision	Since we’re a factory and we could need to provide single services multiple times, this architecture would fit very well in this case. It will also be useful in case of Crash of a datacenter and we will need to deploy our services on the Public Cloud.
	Identifier: Name	Con#2-Opt#2: Monolithic Architecture
	Description	In monolithic architectures, all processes are closely related to each other and run as a single service.
	Status	This option is **rejected**.
	Relationship(s)	
	Rationale of decision	Despite it’s easy to implement, easy to scale, since we have a lot of services, so a large application, it is not going to be efficient, we’re going to instantiate the whole application everytime we need to instantiate even only one service

Concern (Identifier: Description)		Con#2: How should a minimal hardware CAPEX can be achieved?
Ranking criteria (Identifier: Name)		Cr#1: Availability Cr#2: Performance Cr#3: Costs
Options	***Identifier: Name***	Con#2-Opt#1:”RAID” Architecture
	Description	We share the fault tolerance to all datacenters. Strategy based on NAS fault tolerance Server needed: We need 6 servers that run at 83% in each datacenter, distributing the crash of a Datacenter in all the other datacenters of the other factories, which communicates through a WAN.
	Status	This option is **Rejected.**
	Relationship(s)	This decision is **related to** decision Con#1-Opt#1: "Microservice Architecture"
	Evaluation	Cr#1: How do we guarantee the availability of T1 services in case of failure? In Case of a crash of one datacenter, we guarantee it by moving these services to the adjacent datacenter that will take care of these services and share the T2 with the other Datacenters. All Datacenters will bring their percentage of cpu work to 100 Cr#2: We keep always the same performances as we reach in case of a datacenter crash exactly 100% CPU usage in each server of every datacenter for all the factories. Cr#3: The total cost will be 475.200 USD.
	Rationale of decision	The total cost will be 475.200 USD Against the 396.000 USD of the Disaster recovery as a service. In the unlikely case of 13 crashes that last at most a month of datacenters in 3 years, then the RAID will be better than “Disaster recovery as a service” from a cost point of view.
Options	***Identifier: Name***	Con#2-Opt#2: Disaster Recovery as a Service Architecture
	Description	We use 5 servers up to 100% and we use cloud servers only in case of error. We guarantee fault tolerance for T1 services using the adjacent DC and T2 services are sent to the cloud.
	Status	This option is **Accepted.**
	Relationship(s)	This decision is **related to** decision Con#1-Opt#1: "Microservice Architecture"
	Evaluation	Cr#1: How do we guarantee the availability of T1 services in case of failure? In Case of a crash of one datacenter, we guarantee it by moving these services to the adjacent datacenter that will take care of these services and T2 Services will be computed in a Pay as You Use AWS Public Cloud.

		Cr#2: Our servers run every time at 100% CPU, even in case of Failure of an entire datacenter. Cr#3: The total cost will be 396.000 USD + any failure that occurred within 3 years (Calculated above).
	Rationale of decision	The total cost will be 396.00 USD despite the 475.200 USD of the “RAID” architecture. Costs in case of error won’t be high such that we’re going to prefer the RAID one, calculus were made above in the “Strategies analysis”.
Options	***Identifier: Name***	Con#2-Opt#3:“Rent Server for T2 Services” Strategy:
	Description	We only guarantee fault tolerance for T1 services using the adjacent DC and rent cloud servers for T2 services (only the remaining 20%). T1 services make up 40% of the current load. We therefore guarantee 80% of the current load in the two adjacent datacenters and the remaining 20% is rented in the cloud.
	Status	This option is **Rejected.**
	Relationship(s)	This decision is **related to** decision Con#1-Opt#1: "Microservice Architecture"
	Evaluation	Cr#1: We guarantee the availability because the adjacent Datacenter will always be able to recover every T1 services that crashed in the fault Datacenter. Cr#2: Our servers run every time at 100% CPU, even in case of Failure of an entire datacenter, but at the same time we’re always renting AWS Public Cloud servers. Cr#3: The total cost will be 499.957,2 USD + any failure that occurred within 3 years (Calculated above).
	Rationale of decision	The total cost will be 499.957,2 USD despite the 396.200 USD of the Disaster Recovery as a Service Architecture.

Concern (Identifier: Description)		Con#3: What type of Cloud Service do we need?
Ranking criteria (Identifier: Name)		Cr#1: Adaptability Cr#2: Security Cr#3: Scalability of services Cr#4: Costs
Options	***Identifier: Name***	Con#3-Opt#1: Public Cloud Architecture
	Description	A public cloud is a platform that uses the standard cloud computing model to make resources available to users remotely. Public cloud services may be free or offered through a variety of subscription or on-demand pricing schemes, including a pay-per-usage model.
	Status	This option is **rejected**
	Relationship(s)	This decision is in **conflict** with T1 Services security.
	Evaluation	Cr#1: In Public Clouds, resources cannot be adapted precisely by any individual client, as multiple clients use the same computing or resource pool. Cr#2: Public Clouds have a fundamental security compliance model. To counter any shortfalls, many providers offer additional protection to add ons. Cr#3: Public Clouds offer good scalability both vertically and horizontally. This allows us to disregard the hardware needed for our services. Cr#4: In Public Cloud, using the pay-per-usage mode, it is possible to minimize costs by paying only the computing resources actually used.
	Rationale of decision	This option is **rejected** because T1 services may contain valuable and secret information for the company that cannot be trusted to the security of an external company
	Identifier: Name	Con#3-Opt#2: Private Cloud Architecture
	Description	Private cloud is a term for computing services offered over the Internet or a private internal network only to selected users and not to the general public.
	Status	This option is **rejected**.
	Relationship(s)	This decision is **related to** T1 services security.
	Evaluation	Cr#1: Private Clouds allow a computer, network, and storage capacity to alter and adapt to a particular client’s requirements. Thus, it will enable the technology to be adjusted precisely by private parties. Cr#2: All data is saved and managed on servers to which no other company has access. This greatly improves data privacy. If the servers are on-site, they are managed by an internal IT team. Cr#3: Scalability in private clouds is in-house. Cr#4: Private cloud solutions require substantial upfront investments to implement massive Software, hardware, and staffing requirements. On-going costs also include growth and maintenance costs.

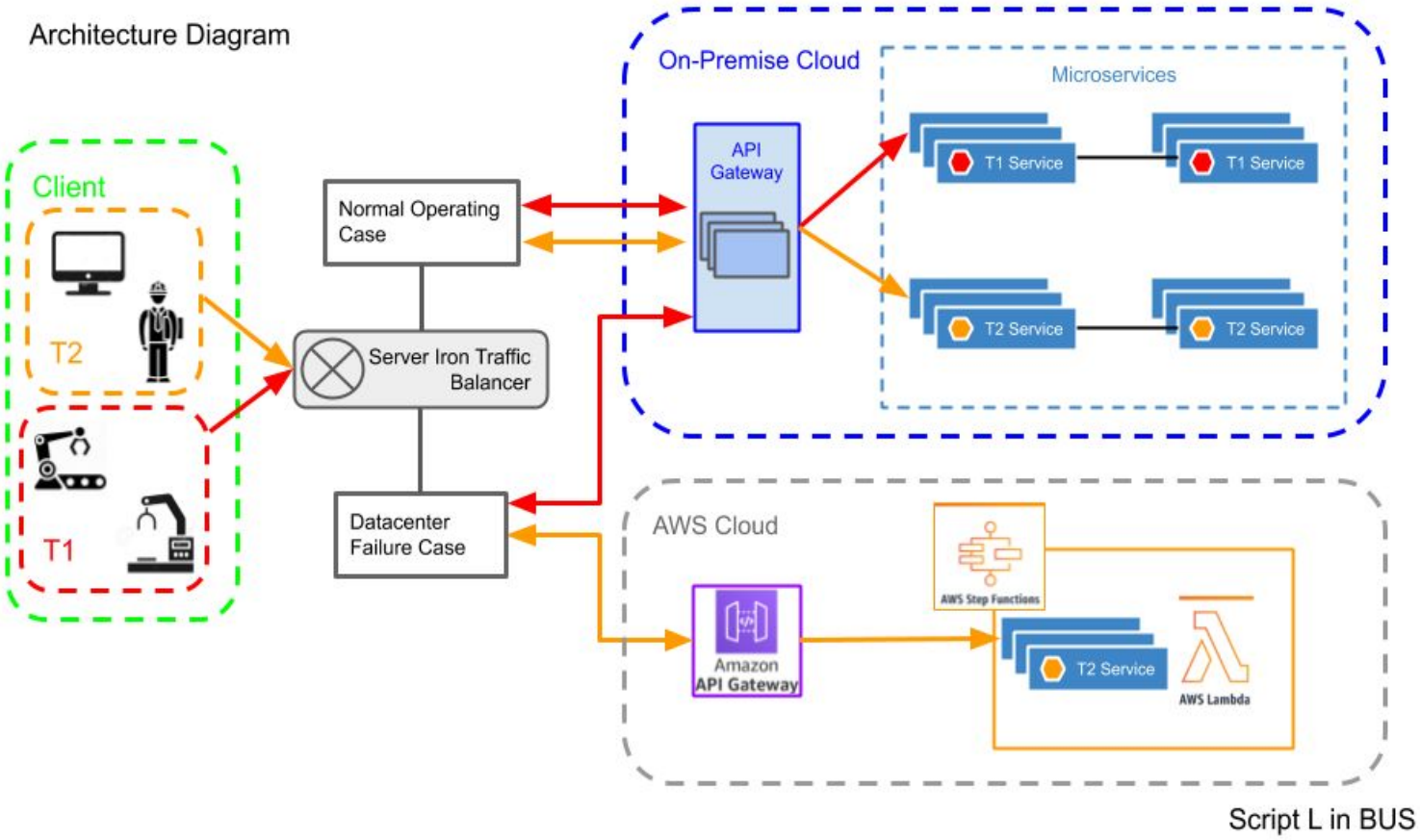
	Rationale of decision	Private cloud respects all criteria and it is more security than public cloud even if it is limited to the hardware available, we refuse a full private-cloud because we need also public cloud for our strategy.
	Identifier: Name	Con#3-Opt#3: Hybrid Cloud Architecture
	Description	Hybrid cloud is a solution that combines a private cloud with one or more public cloud services, with proprietary software that enables communication between each service. Hybrid cloud services are extremely powerful as they give businesses more control over their private data.
	Status	This option is **decided**
	Relationship(s)	This decision **synergizes** with Con#1-Opt#1: “Microservices architecture”
	Evaluation	Cr#1: Adaptability: There is no problem of adaptability since it is a combination of both private and public cloud, it has both the advantages of them too. Cr#2: Security: Almost unlimited due to on-demand cloud resources, guaranteed also by the usage of the private (side) cloud. Cr#3: Scalability of services: Mixed with microservices we’re guaranteed scalability for any amount of incoming demand. We are going to Expand and reduce our work as we need it at a certain moment. Cr#4: Costs: Lower capital expenditure (capex): In our architecture we are going to use a Hybrid cloud only in the case we’re going to lose a Datacenter. This is because after a few calculations that are written above in the “Strategies analysis” we saw that using public cloud to recover T2 Services in Cloud as in our decided Architecture is the most efficient than all the other strategies we have studied.
	Rationale of decision	It is a good solution because it contains public cloud and private cloud and so also respect all the criteria. In particolar, if there is a normal operating T1 and T2 service running On-Premise Cloud while when a Datacenter crashes we need a Public Cloud for T2 services.

Concern (Identifier: Description)		Con#4: How do we manage the services?
Ranking criteria (Identifier: Name)		Cr#1: Containerization tool Cr#2: Container orchestrator Cr#3: Services communication
Options	***Identifier: Name***	Con#4-Opt#1: Docker containerization tool
	Description	Docker is an Open-Source tool that allows containerizing applications and easily deploy them.
	Status	This option is **decided**
	Relationship(s)	This decision **is related to** decision Con#1-Opt#1: "Microservice Architecture" This decision **is related** to decision Con#3-Opt#3: ”Hybrid Cloud Architecture”
	Evaluation	Cr#1: We will use Docker in order to containerize our services (both T1 and T2). It will be crucial in order to orchestrate and generally manage them efficiently.
	Rationale of decision	We decided to use Docker because containers include the minimal runtime requirements of the application, reducing their size and allowing them to be deployed quickly, and an application and all its dependencies can be bundled into a single container that is independent from the host version of Linux kernel, platform distribution, or deployment model, gaining portability across other machines (in our case the Cloud Service platform). This container can be transferred to another machine that runs **Docker**, and executed there without compatibility issues. Docker also reduces effort and risk of problems with application dependencies, reducing maintenance for our services.
	Identifier: Name	Con#4-opt#2: Kubernetes orchestrator
	Description	Kubernetes is a container orchestrator and manager. It works with a lot of containerization systems, including Docker.
	Status	This option is **decided**
	Relationship(s)	This decision **is related** to decision Con#4-Opt#1: “Docker containerization tool”
	Evaluation	Cr#2: We will use Kubernetes to orchestrate the containers deployed with Docker
	Rationale of decision	Kubernetes will be useful because we will just need to provide our desired state to our clusters and it will keep our services up, scaling them if needed, just communicating with the kubelets.
	Identifier: Name	Con#4-Opt-3: Kafka
	Description	Kafka is an Open-Source stream processing platform that will be useful to make our services communicate.
	Status	This option is **decided.**
	Relationship(s)	This decision **is related** to decision Con#1-Opt#1: “Microservice architecture”
	Evaluation	Cr#3: We will use Kafka for service communications.
	Rationale of decision	Kafka is decided for message broker

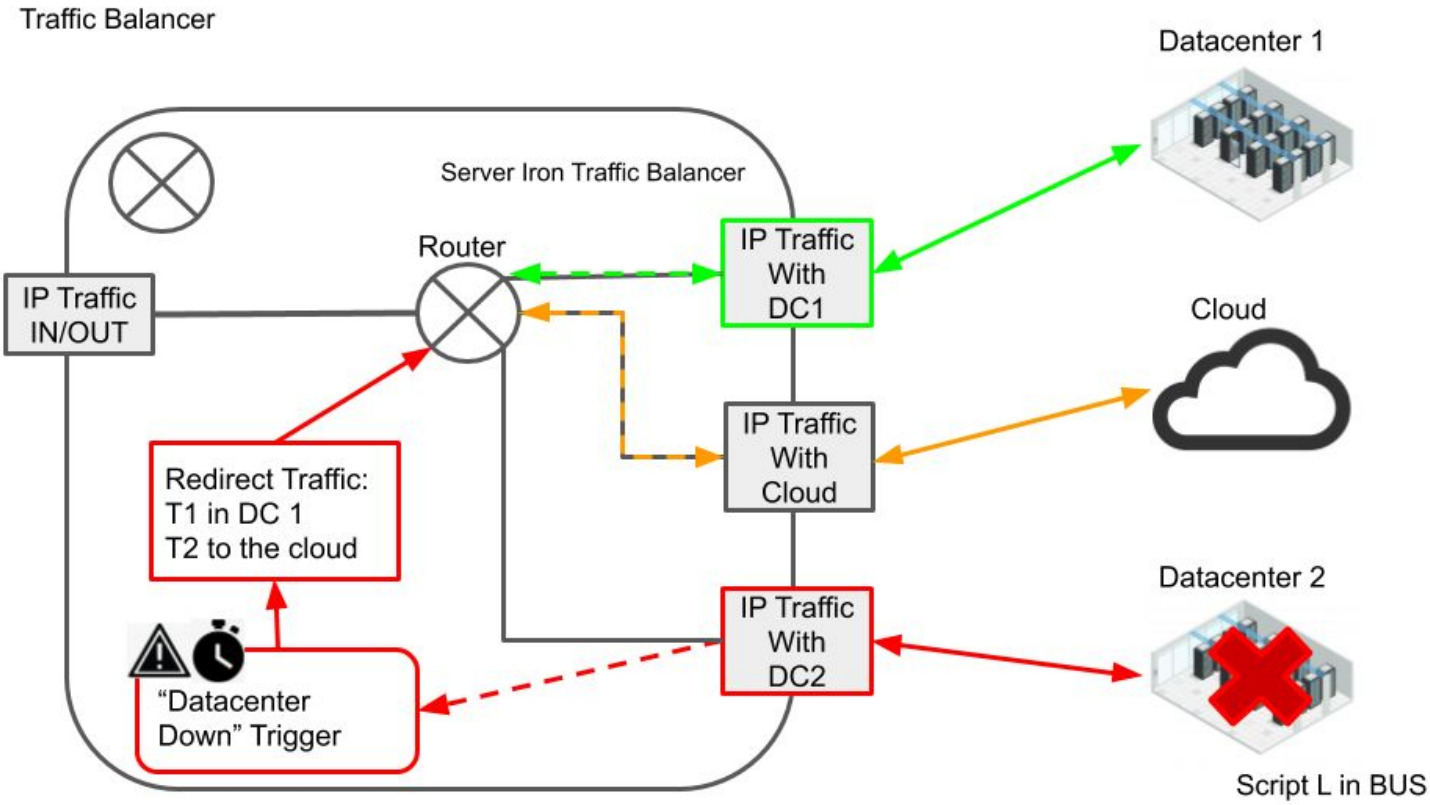
Concern (Identifier: Description)		**Con#5: Which deployment models for cloud services should be used?**
Ranking criteria (Identifier: Name)		Cr#1: Infrastructure management Cr#2: Product or service development Cr#3: Usage.
Options	***Identifier: Name***	Con#5-Opt#1: Software as a Service (SaaS)
	Description	It offers the end user a complete solution in which a fee is paid for the use of software whose interface is easily accessible on the client side.
	Status	This option is **Rejected.**
	Relationship(s)	-
	Evaluation	Cr#1: We should have our system environment under our control. Cr#2: We could need to develop our software for our purposes. Cr#3: No usage found in our case.
	Rationale of decision	This option was not chosen because using the Hybrid Cloud we chose other infrastructures because it is not very efficiently in terms of modelling environment
	Identifier: Name	Con#5-Opt#2: Platform as a service (PaaS)
	Description	In addition to the basic infrastructure resources, a series of pre-installed components and software can be rented, to be used for the development of a product or service. In short, additional services, which can be of different types: operating system, database, applications or middleware.
	Status	This option is **decided**
	Relationship(s)	Strongly **related to** Con#3-Opt#3: “Hybrid Cloud Architecture”
	Evaluation	Cr#1: We’re going to use AWS Public Cloud. Cr#2: We will containerize and deploy our services and Lambda Tools will allow us to do it easily and efficiently. Cr#3: We’re going to use Platform as a Service for Pay as You Use Public Cloud .
	Rationale of decision	For the public part the infrastructure is chosen by the provider and the private part are our datacenters. Moreover, the management of infrastructure is entrusted to the company that leases the infrastructure.
	Identifier: Name	Con#5-Opt#3: Infrastructure as a service (IaaS)
	Description	Is the first level of cloud, used for accessing, monitoring and managing remote infrastructures, such as virtualized computers, storage, networking and network services
	Status	This option is **decided**
	Relationship(s)	This decision is **related to** Con#3-Opt#3: “Hybrid Cloud Architecture”
	Evaluation	Cr#1: We’re going to use the Open-Shift platform on-premise cloud. Cr#2: We will containerize and deploy our services and Open-Shift will allow use to do it easily and efficiently. Cr#3: We’re going to use Platform as a Service for on-premise Cloud.
	Rationale of decision	OpenShift accelerates application development by including the tools that companies need to be agile and efficient. With OpenShift, it is possible to deploy applications

		quickly, become less siloed, be more interactive, and increase collaboration.

Architecture Diagrams



In this part of the architecture diagram we have an overview of the whole architecture. We start from the clients that are composed by: Machines, Users (managers, employees). The request that we send/receive from the services travel through the ServerIron Traffic Balancer; API Gateways are used to reach the microservices that are containerized with Docker and Orchestrated by Kubernetes. That’s how the system should operate normally. In case of failure of an entire Datacenter, we switch to the Hybrid side of this architecture, moving the remaining T2 Services on the AWS Public Cloud; leaving to the adjacent online Datacenter all the T1 services and few T2 to fill the spare space.



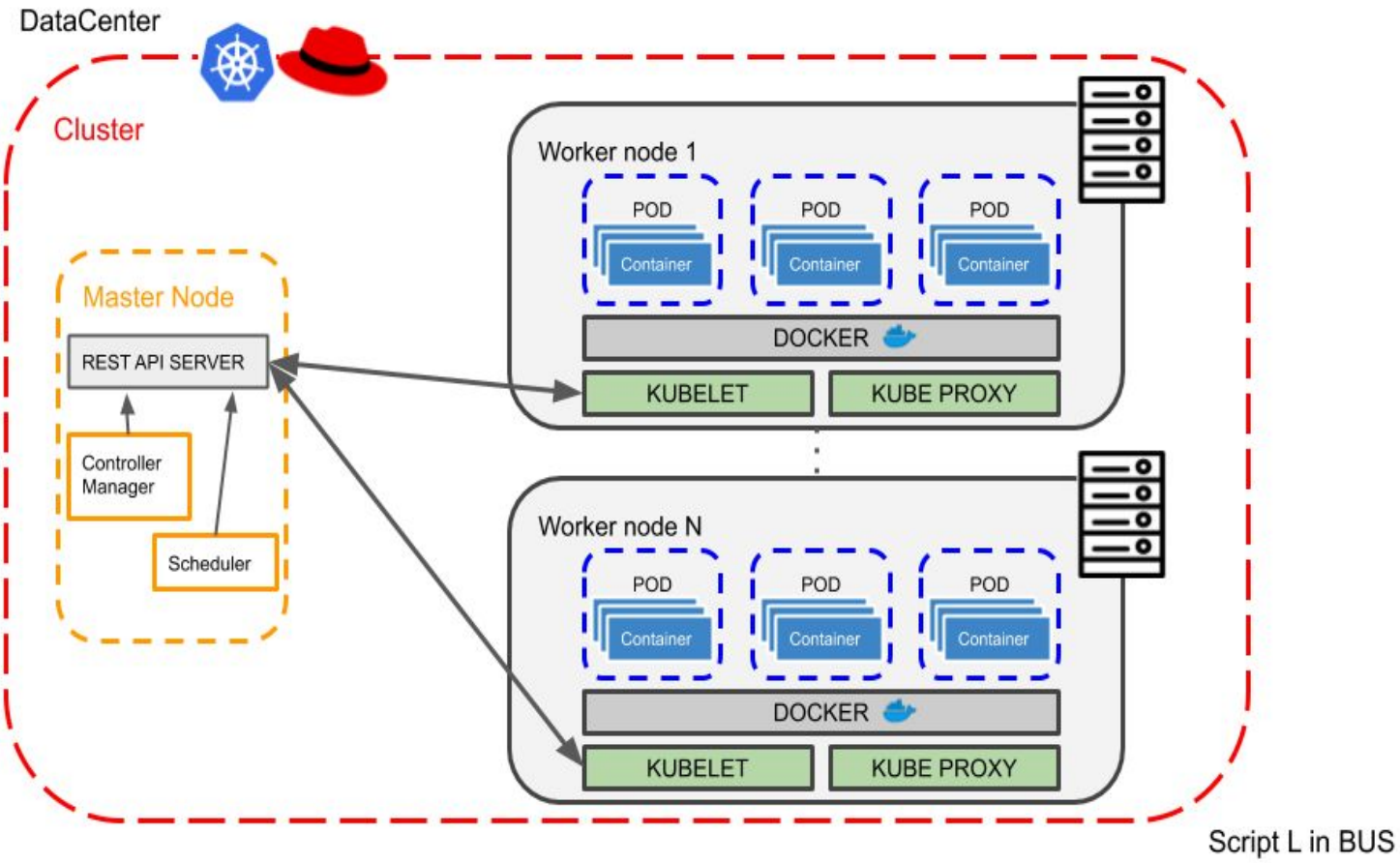
We focus on how Communication is done between client/server and how Disaster Recovery is done. We send data through IP protocol, that reaches a routing point that routes the requests to a suitable Datacenter; if one Datacenter goes down, we have a Trigger that notifies the Router, which will be forwarding incoming requests to the remaining datacenter (for every T1 Service and 20% of T2) and the AWS public cloud (strictly only for T2 Services, 100% remaining).

We assume that services are recognizable since their names are composed this way:

T1: t1_*servicename*

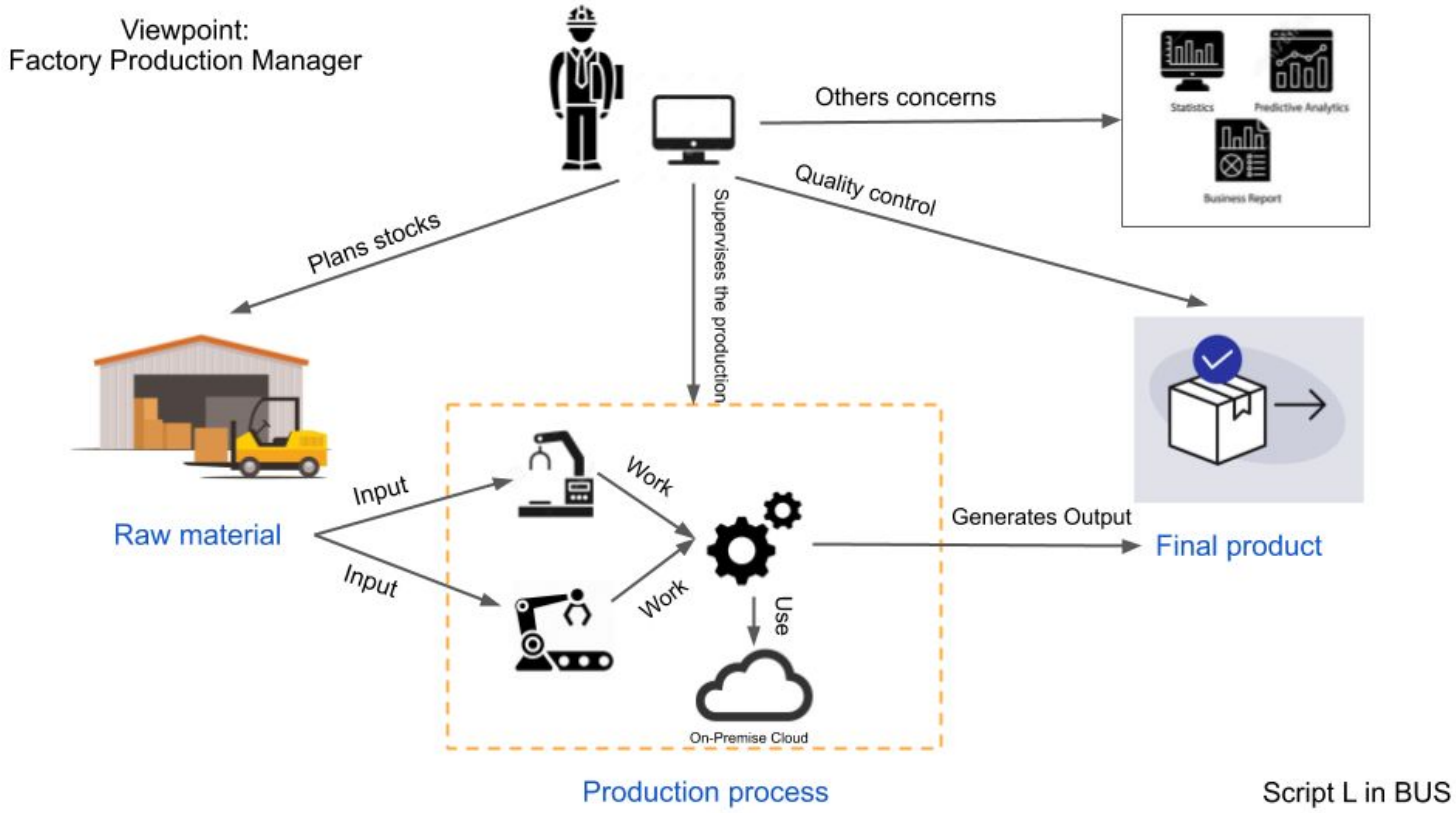
T2: t2_*servicename*

The router will be able to recognize if a service is T1 or T2 and forward them correctly to the datacenter or Public Cloud.



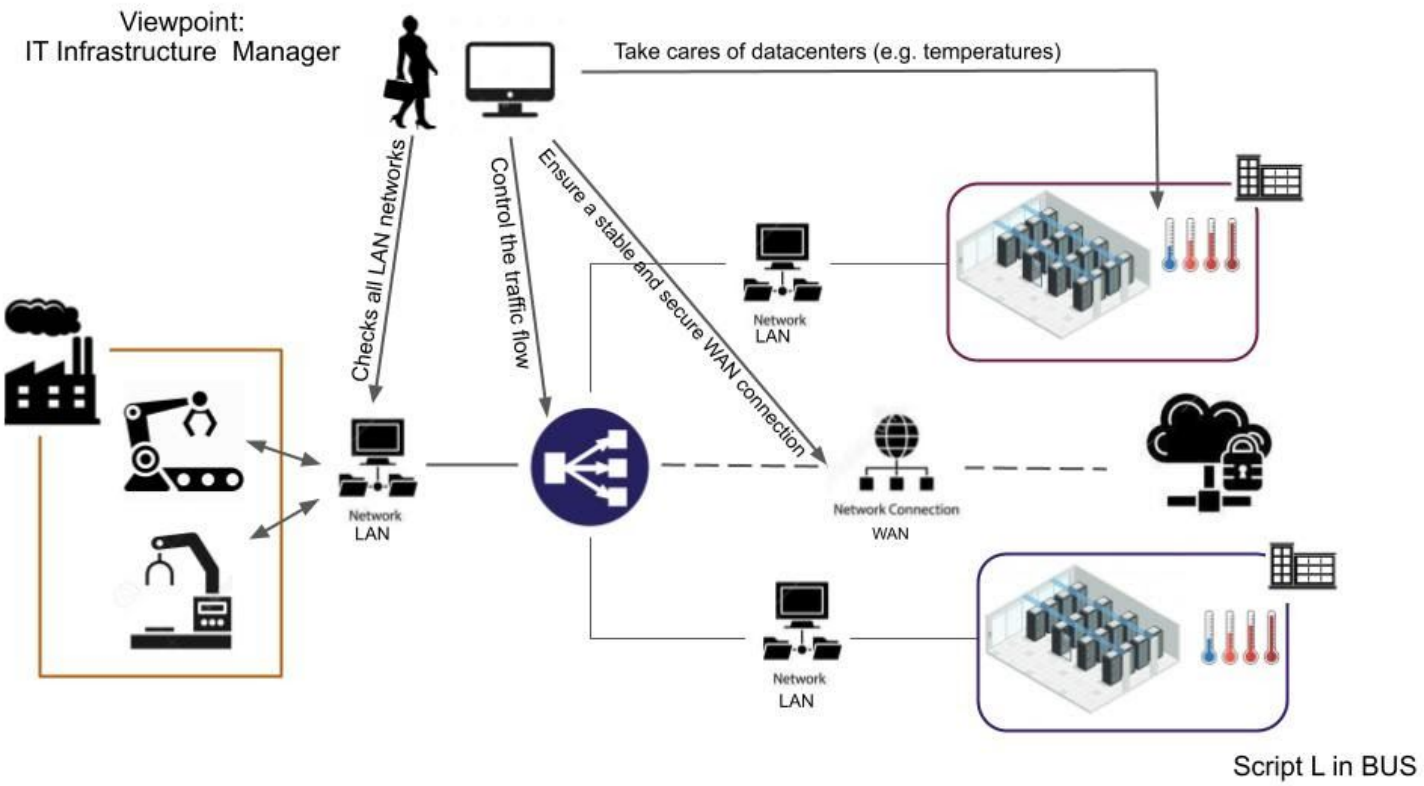
In the last part we represent how each datacenter is composed and how a service is. In this picture we represented how the Master Node of Kubernetes is connected to the Kubelets, considering that we feed the Master Node with our desired state. Worker nodes run services that are containerized with Docker. They may run multiple replicas of different Pods. Pods contain the docker images of the services.

Viewpoints



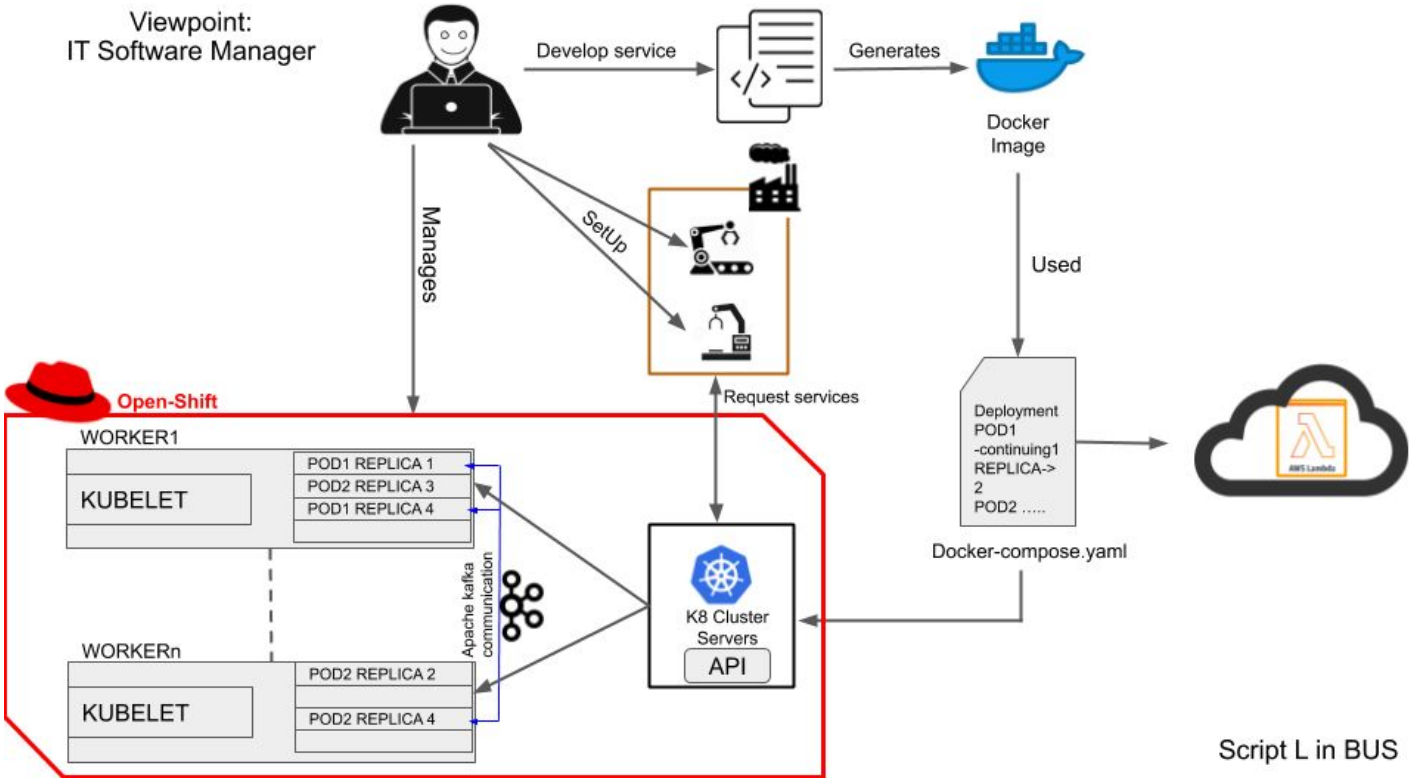
In this image we represent the main concerns of the Factory Production Manager.
In particolar all the concerns of the factory production manager are:

- Controls and manages what happens within the production departments.
- Knows the machinery responsible for creating a product. It is your responsibility to act in case of breakdowns or malfunctions.
- Plan stocks and manage the warehouse. Checks the procurement of materials and compliance with the budget.
- Knows how to use software related to production.
- Establishes and plans the calendar of activities pertaining to the various production and logistic departments of the company.
- Optimizes all aspects related to production, such as timing and methods that are followed or put into practice.
- Plans strategies that will be presented to the executives.
- Studies the human resources employed at the company and places them where there is greater efficiency.
- Facilitates communication between the different departments in order to speed up the supply chain.



Now represent the main concerns of the IT Infrastructure Manager.
It is the duty of the Infrastructure Manager to keep the entire infrastructure of the factory under control and in particular:

- Monitors and ensures the correct functioning of the LAN as well as the WAN.
- Check the ServerIron traffic flow
- It constantly monitors the status of the datacenters such as their temperature.



The last viewpoint represents IT Software Manager and his main concerns.
In particolar all concerns of IT Software Manager are:

- Design effective strategies and plans to overcome global competition.
- Replace legacy systems and infrastructure with modern ones.
- Adopt new technologies to reduce time-to-market.
- Develop the software and services involved in the factory.
- Outsourcing or in-sourcing the software development process.
- Hire people with proper skill sets to upkeep in-house tech processes.
- Generate ROI on a consistent basis.
- Sets up the whole software system adapted to the infrastructure.

We can summarize the main concerns with the following table:

Managers/Concerns	Factory Production Manager	IT Infrastructure Manager	IT Software Manager
Reliability	X	X	X
Continuous Production	X		
Safety		X	X
Management Team	X		
Maintenance	X	X	X
Installation		X	
Performances	X	X	X
Networking		X	
Costs	X	X	

Architectural Significant Requirement

- **Maximizing hardware CPU utilization, to minimize hardware Capex:**

  In Case of a crash of one datacenter, we guarantee T1 availability by moving these services to the adjacent datacenter that will take care of these services and T2 Services will be computed in a Pay as You Use AWS Public Cloud; so we could reduce the number of servers used per each datacenter to 5 and the servers always run at 100% even in case of crash of an entire datacenter. The reduction of servers also brought us less costs.

- **T1 services have 99.999% availability:**

  Yes, they always run locally even if one datacenter goes down. They will always be running locally in our Datacenter, in the case of a crash, one datacenter will be able to run all the T1 Services until the second datacenter comes back up. The ServerIron Traffic Balancer instantaneously detects the failure of a Datacenter, the “Datacenter down” will be triggered and the Traffic Balancer will be immediately forwarding T2 to the Public Cloud.

- **In case of a major failure, T2 services can operate at degraded performance:**

  Yes, most of them (100% because the total is 120%) will run on Public Cloud so they will be run but of course with more latency so at degraded performance.

- **T2 services have a MTTR < 2 hours:**

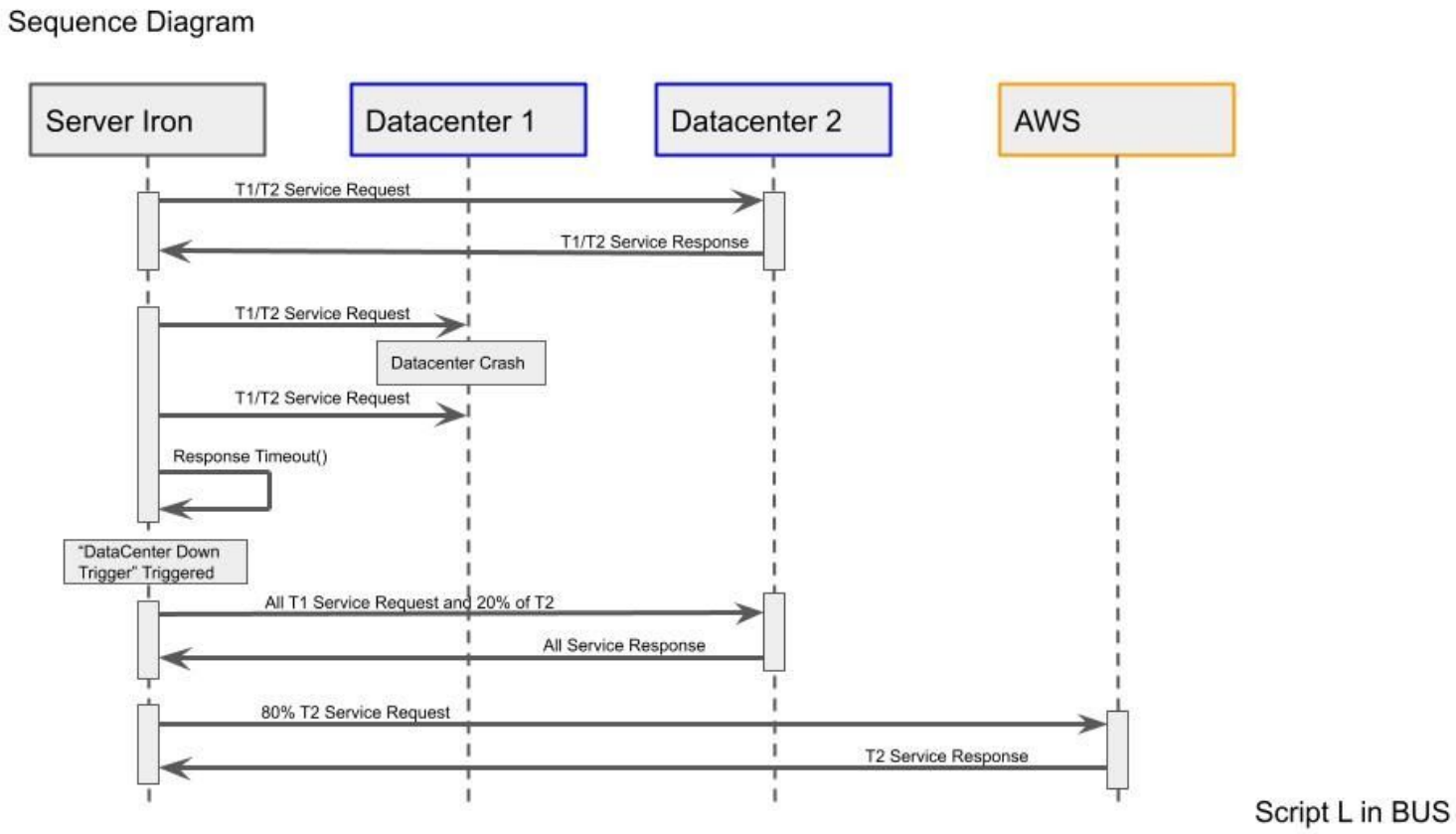
  They are instantaneously recovered in Public Cloud if a Datacenter goes down.

- **The architecture is compliant with the following technology roadmap :**

  Docker, Kubernetes, Open Shift, Kafka.

Sequence Diagram

Here we show how we handle fault tolerance in our architecture.



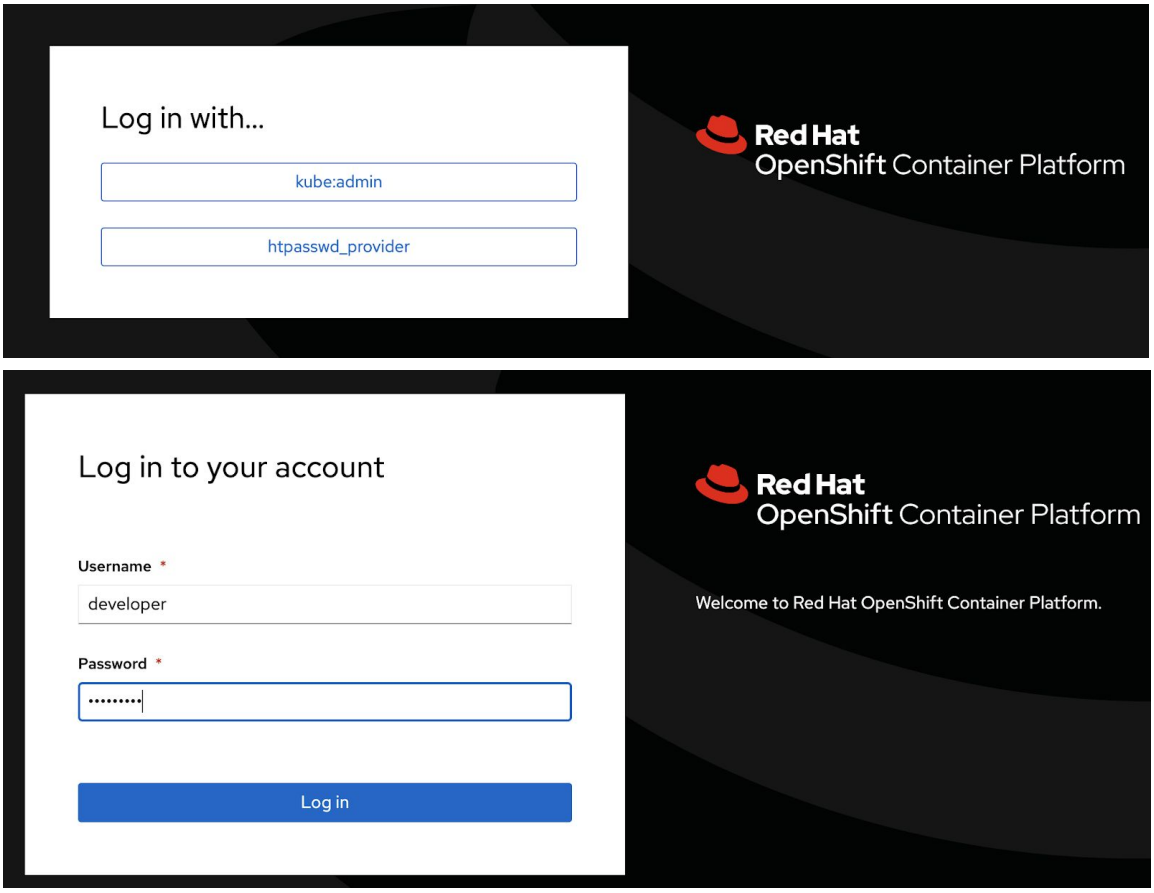
Prototype and Live Demo

In order to implement the architecture we made use of Openshift free “developer” version.

To connect to Openshift we execute the following commands through the CLI:

```
laura@LARUSs-MacBook-Pro-2 ~ % crc start
WARN A new version (1.22.0) has been published on https://cloud.redhat.com/openshift/install/crc/installer-provisioned
INFO Checking if running as non-root
INFO Checking if podman remote executable is cached
INFO Checking if admin-helper executable is cached
INFO Checking minimum RAM requirements
INFO Checking if HyperKit is installed
INFO Checking if crc-driver-hyperkit is installed
INFO Checking file permissions for /etc/hosts
INFO Checking file permissions for /etc/resolver/testing
INFO Starting CodeReady Containers VM for OpenShift 4.6.9...
INFO CodeReady Containers VM is running
INFO Starting network time synchronization in CodeReady Containers VM
INFO Network restart not needed
INFO Check internal and public DNS query ...
INFO Check DNS query from host ...
INFO Verifying validity of the kubelet certificates ...
INFO Starting OpenShift kubelet service
INFO Starting OpenShift cluster ... [waiting 3m]
INFO Updating kubeconfig
WARN The cluster might report a degraded or error state. This is expected since several operators have been disabled t
 lower the resource usage. For more information, please consult the documentation


Started the OpenShift cluster

To access the cluster, first set up your environment by following the instructions returned by executing 'crc oc-env'.
Then you can access your cluster by running 'oc login -u developer -p developer https://api.crc.testing:6443'.
To login as a cluster admin, run 'oc login -u kubeadmin -p D4tTL-hJYdy-qe36Y-okTwh https://api.crc.testing:6443'.

You can also run 'crc console' and use the above credentials to access the OpenShift web console.
The console will open in your default browser.
laura@LARUSs-MacBook-Pro-2 ~ % crc console
Opening the OpenShift Web Console in the default browser...
laura@LARUSs-MacBook-Pro-2 ~ %
```

Here’s an image of openshift login page:



After creating the deployment and services (.yaml) files we connect through Visual Studio Code CLI to openshift’s command line with the following commands:

```
LARUSs-MacBook-Pro-2:serveriron laura$ oc login
Authentication required for https://api.crc.testing:6443 (openshift)
Username: developer
Password:
Login successful.

You have one project on this server: "sa-2021"

Using project "sa-2021".
LARUSs-MacBook-Pro-2:serveriron laura$
```

After logging in we can deploy the dc1.yaml and dc2.yaml files and expose them with dc1-s.yaml and dc2-s.yaml always through the CLI this way:

```
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
LARUSs-MacBook-Pro-2:serveriron laura$ kubectl apply -f dc1.yaml
LARUSs-MacBook-Pro-2:serveriron laura$ kubectl apply -f dc1-s.yaml
```

After this we can start to delete or create replicas for our pods in this way:

```
LARUSs-MacBook-Pro-2:serveriron laura$ kubectl scale deployment dc1 --replicas=5
```

Here is an example for scale pods.

Now we start serverironTest.js that will simulate the ServerIron described in our project. In our prototype this server will be sending request every 5 seconds to the Datacenters and check that they are up.
Below pieces of the source code of serverironTest.js:

```
var timer = setInterval(function() {


if(datacenterOnLine){
  var randomico = Math.floor(Math.random() * 100) + 1;
  request(urldc1 + 'T1/'+randomico, { json: true }, (err, res, body) => {
    if (res.statusCode==503) {
      datacenterOnLine = false;
      console.log("Datacenter 1 T1 : Offline")
```

```
  }
  else{
    console.log("Datacenter 1 T1 SERVICE : "+res.body);
  }
  });

request(urldc1 + 'T2', { json: true }, (err, res, body) => {
if (res.statusCode==503) {
  console.log("Datacenter 1 T2 : Offline")

}
else{
  console.log("Datacenter 1 T2 SERVICE : "+res.body);
}
});
var randomico2 = Math.floor(Math.random() * 100) + 1;

request(urldc2+'T1/'+randomico2, { json: true }, (err, res, body) => {
  if (res.statusCode==503) {
    datacenterOnLine = false;
    console.log("Datacenter 2 T1 : Offline")

  }
  else{
    console.log("Datacenter 2 T1 SERVICE : "+res.body);
  }
  });

request(urldc2+'T2', { json: true }, (err, res, body) => {
if (res.statusCode==503) {
  console.log("Datacenter 2 T2:Offline")

}
else{
  console.log("Datacenter 2 T2 SERVICE : "+res.body);
}
});
```

We simulated the case where Datacenter 1 will go offline decreasing the number of pods to 0 with command:

- **kubectl scale deployment dc1 --replicas=0**

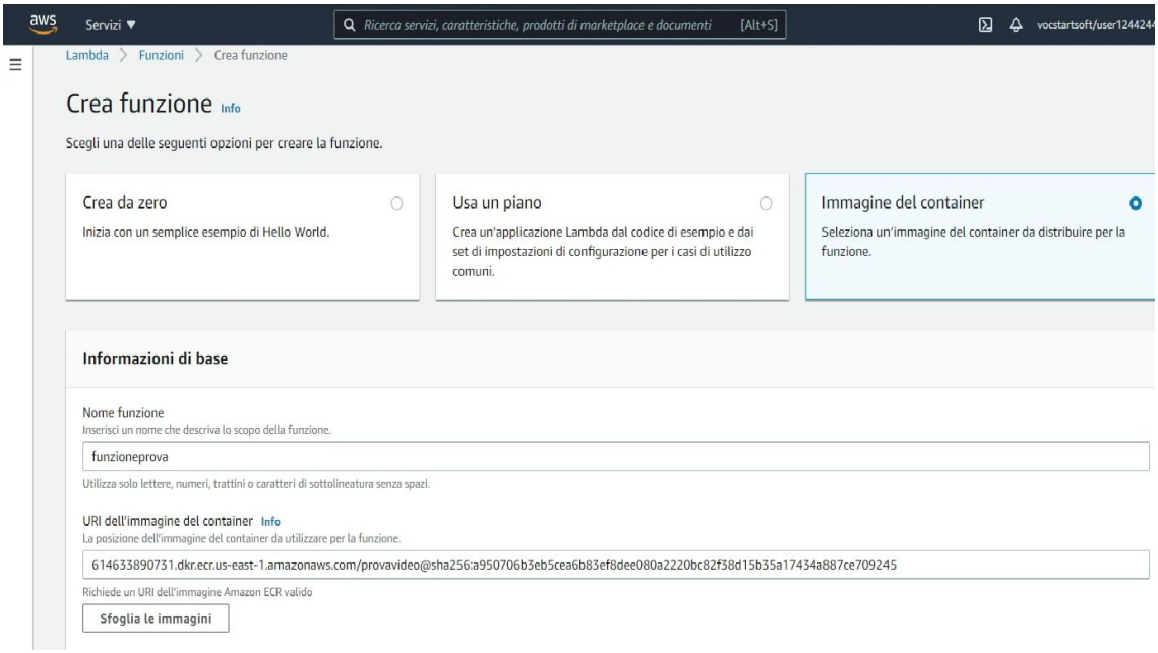
serveironTest.js will manage to recognize the crash and put the T1 in the Datacenter 2 and the T2 in AWS.

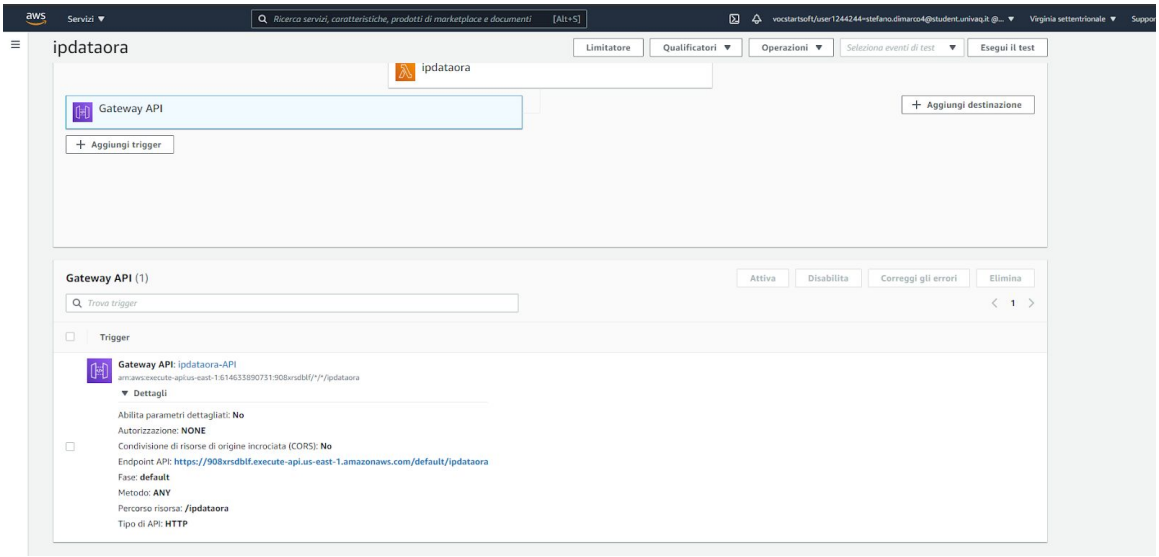
Here is a part of the code regarding the error handling. In the first part of the code we modeled the case where the Datacenter 2 takes the T1 services of Datacenter 1 while on the second part we request the T2 Services to AWS Lambda in pay as you use mode using the API Gateway of the function interested (in our case the API gateway of ipdataora function).

```
    request( urldc2+'T1/'+randomico2, { json: true }, (err, res, body) => {
      if (res.statusCode==503) {
        datacenterOnLine = false;
        console.log("Datacenter 2 T1 : Offline")

      }
      else{
        console.log("Datacenter 2 T1 SERVICE : "+res.body);
      }
      });
```

```
request('https://908xrsdblf.execute-api.us-east-1.amazonaws.com/default/ipdataora', { json: true }, (err, res, body) => {
    if (err) {
      console.log("AWS1 : Offline")

    }
    else{

      console.log("AWS LAMBDA PER T2 SERVICE : "+res.body);

    }
  });
```

For the Public Cloud part we used AWS lambda functions in order to deploy our T2 Services when needed.
We created a lambda function deploying our T2 Service that returns Ip, current date and time:





Source code in github:
serveironTest and yaml files: https://github.com/Laudie/projectSA-2021
T1 and T2 services: https://github.com/Fabgen5/SA2021
AWS lambda: https://github.com/stefdm97/sa2021

Script L in BUS