Faculty of Engineering & Applied Science

SOFE4790U – Distributed Systems

Group Work

Homework – Gateways

Due Date: 11/06/2022

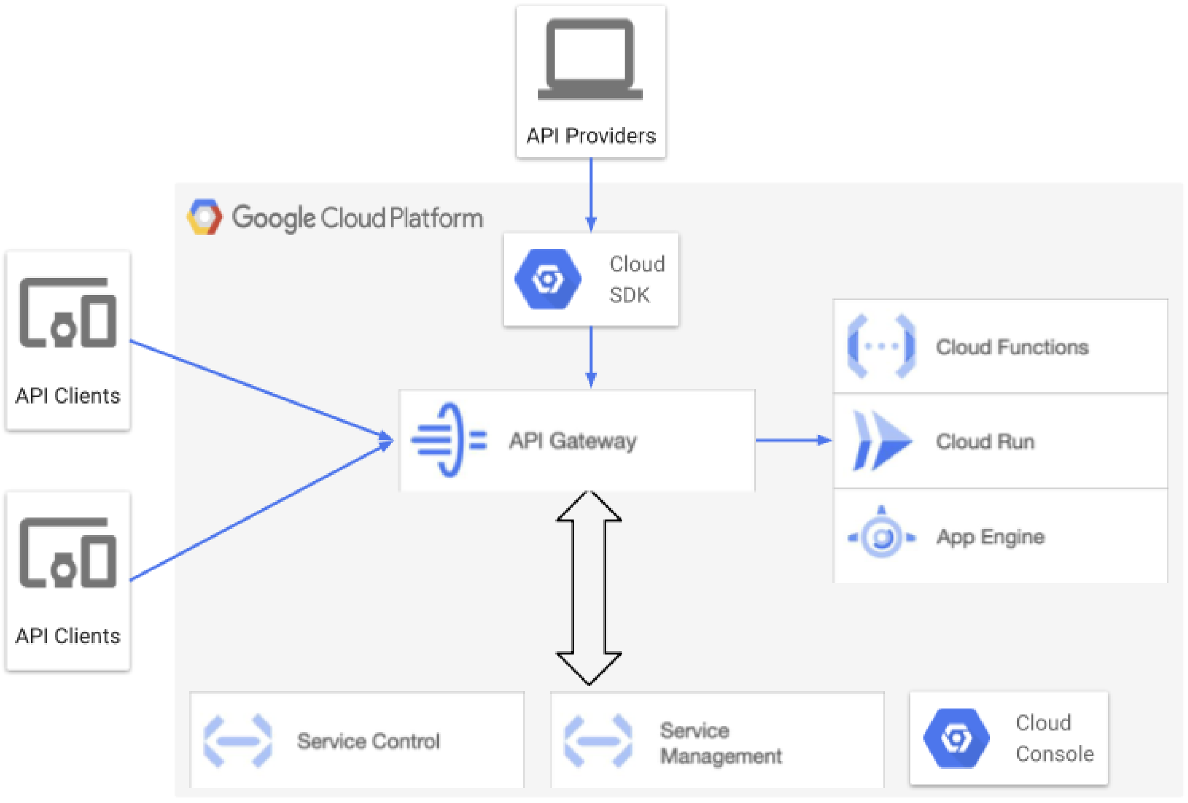
First Name	Last Name	Student ID
Abdul	Bhutta	100785884
Alexander	Campbell	100703650
Mamun	Hossain	100553073
Owen	Musselman	100657709

The idea of gateway is central to a distributed system. Please read the following articles and try to come with a descriptions for; The requirements of a gateway, and a high level design of the various components needed for a gateway to operate.

Gateway Components

Components	Description
Gateway	Managing the services
Control	Applying management rules
Management	Managing and configuration
Console	Deploy, manage, and monitor

An example of a high-level design or architecture for a gateway is shown below, which is used to implement a Google API Gateway. The main components are the gateway API, service control, service management, google cloud CLI, and cloud console.



Gateway Requirements

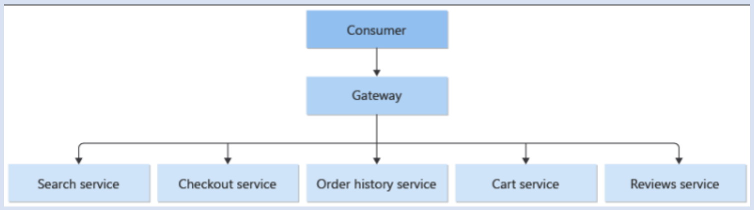
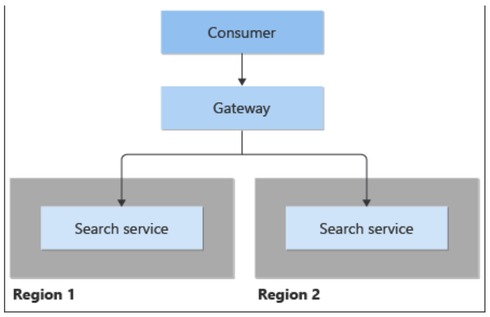
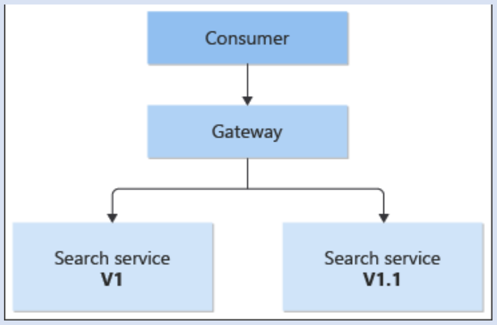
Requirement	Description
Frontend IP	Ip address to access the gateway
Port	To identify and listen to the request
Certificate	Provides security and privacy
Listeners	Used to listen to the request and logically send the request to the backend services.
Protocols	A type of communication among the components. (HTTP, HTTPS)

Requirements for Gateways
● *Reliable*: Gateway needs to be reliable, sending requests to the proper services, and being able to function correctly with heavy traffic loads. ● *Scalable*: Gateways can be scaled up and down with the traffic coming in, making for a more efficient system. ● *Availability*: Gateway should have a high availability to ensure clients can use the system at all times. ● *Performance*: The gateways should perform their tasks (Routing, Aggregation, Offloading) in an acceptable time. ● *Life-Time Costs*: The implementation of the gateway should have a lower cost when it is working for the system. ● *Reusability*: Reusability should be high for the implementation of the gateways, as this can be used in many other systems, reducing design and implementation times for future systems. ● *Maintainability*: The gateway's maintainability should be high, as in the case of a failure in a gateway it should be easy to mend the problem and get it back up and running quickly.

Gateway	**Description**
Gateway Routing	Route the various requests coming into various services into a single endpoint.
Gateway Offloading	Push shared service functionalities to a gateway proxy. Where it moves shared functionalities from applications to the gateway.
Gateway Aggregation	The gateway combines multiple incoming requests into a single request. Better when the client makes many requests to various backend systems.

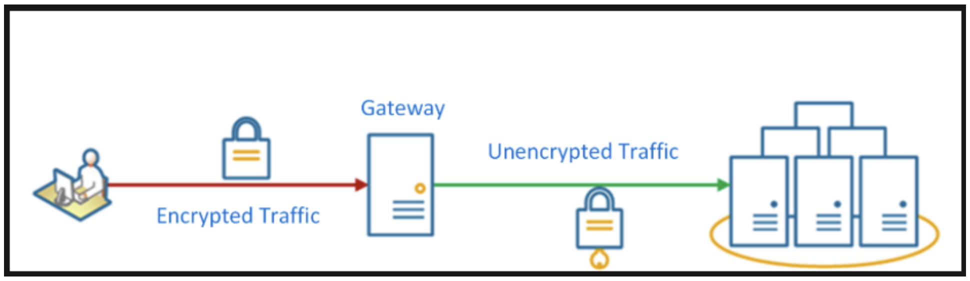
Gateway Routing Pattern

The concern is raised when multiple services, multiple instances of the same service, or multiple versions of the same service are updated, the client will be required to be updated as well. The solution to these concerns is deploying a gateway routing pattern which places a gateway at the endpoint and uses the application layer to route the traffic. This pattern allows the client to communicate with a single endpoint and hides the database or backend infrastructure. Examples of gateway routing patterns integrated within the applications are provided below.

Scenarios	Gateway Implentation
Multiple Services	 Consumer Gateway Search service Checkout service Order history service Cart service Reviews service
Multiple Instances of the the same servces	 Consumer Gateway Search service Search service Region 1 Region 2
Multiple Versions	 Consumer Gateway Search service V1 Search service V1.1

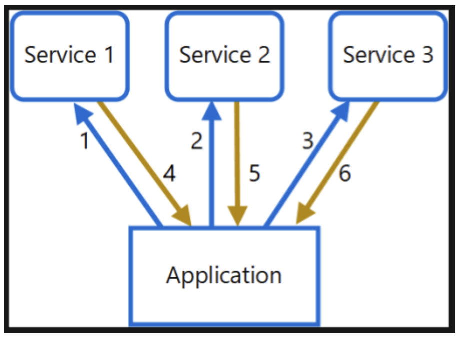
Gateway Offloading Pattern

As there are many applications which require services to be shared and deploying a service in each application causes overhead or deployments error. Some examples of services are security, authentication, and monitoring. Implementing each service with the application is complex, and the gateway offloading pattern is used to solve this issue. The solution is to add the services within the gateway and allowing to simplify the deployments. An example of a gateway offloading pattern is shown below.

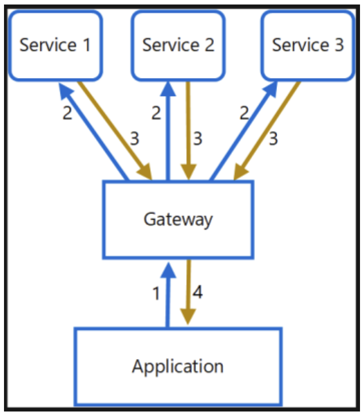


Gateway Aggregation Pattern

The gateway aggregation pattern is applied within applications which require communication with multiple requests to different databases or backend systems. When the client is sending multiple requests to the database or when it requires many services to perform one functionality, it will require the use of many resources, which will impact the performance and scalability of the application. An example of provided below where the application is requesting 3 services over a cellular network. The cellular network is unreliable, inefficient and has high latency, which may result in failure due to connectivity issues.



The solution is to implement a gateway in the middle to reduce the communication between the application and backend services. In the figure below, the gateway will receive the requests from the application, and the gateway will forward the requests to the services. The services will send back a response, and the gateway will combine the responses into a single response and forward it to the application.



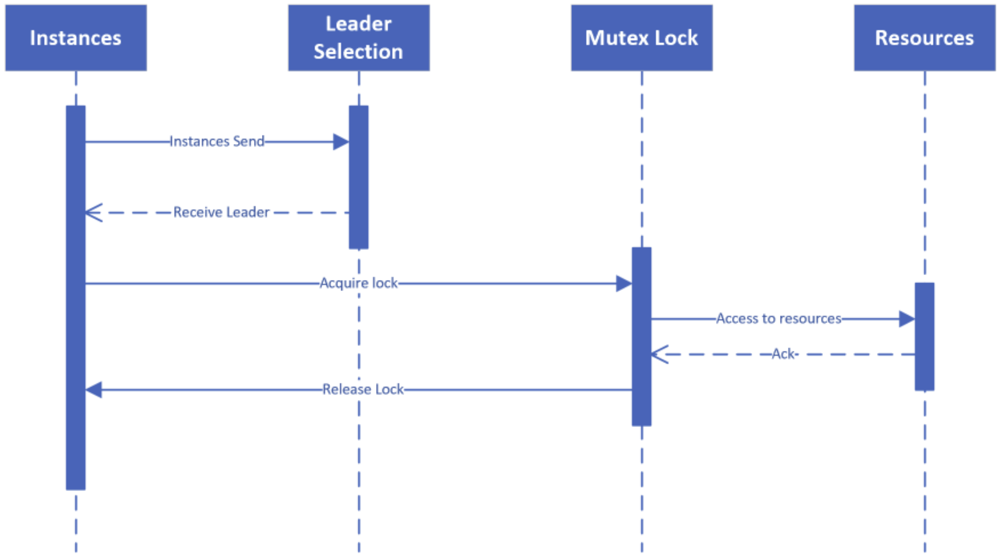
Because gateways are of such importance, they are typically using a clustering technique, where multiple instances of the gateways are deployed. Why would you do that? A technique to get the cluster to operate is to use the Leader Election Pattern.

The reason why you would have multiple instances that insinuate a clustering technique is because you can enable many high-availability clusters of gateway installations, that will help the client and the services use data resources from different cloud services that can help. With these cloud services (eg. Power BI), because gateways are so important, clusters are used to avoid any singular failure point when accessing any on-premise data resource. The gateway cloud service will be used in the primary gateway in a cluster unless the gateway isn't able to be used, which then allows the service to access the next available gateway in the cluster. Gateways use a clustering technique to allow redundancy, load balancing, and failover. This allows the use of another gateway in case one fails. The incoming requests are forwarded to the main gateway of the cluster, allowing for better performance and scalability.

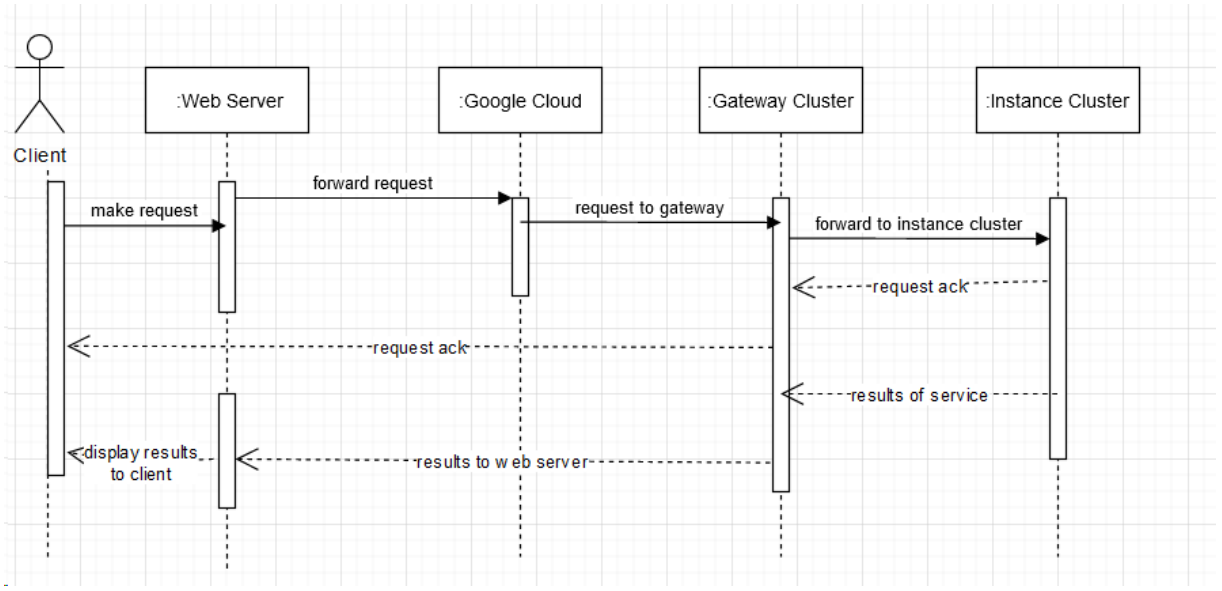
Leader Election Pattern

The Leader Election Pattern is a method to reduce conflict and issues with shared resources in a distributed system through the election of one deployed instance as the leader. When there is a leader instance to coordinate the actions of other deployments, we can avoid issues such as writing data to the same entry at the same time. The main difficulty in implementing this pattern is the required leader election system must be robust and still work despite any failures. This pattern supports concurrency transparency as it aids in several processes being able to run concurrently across a distributed system without causing conflict.

Sequence Diagram for Leader Election Algorithm



Sequence Diagram for Gateway and multiple Instances



========================= END OF GROUP WORK ========================

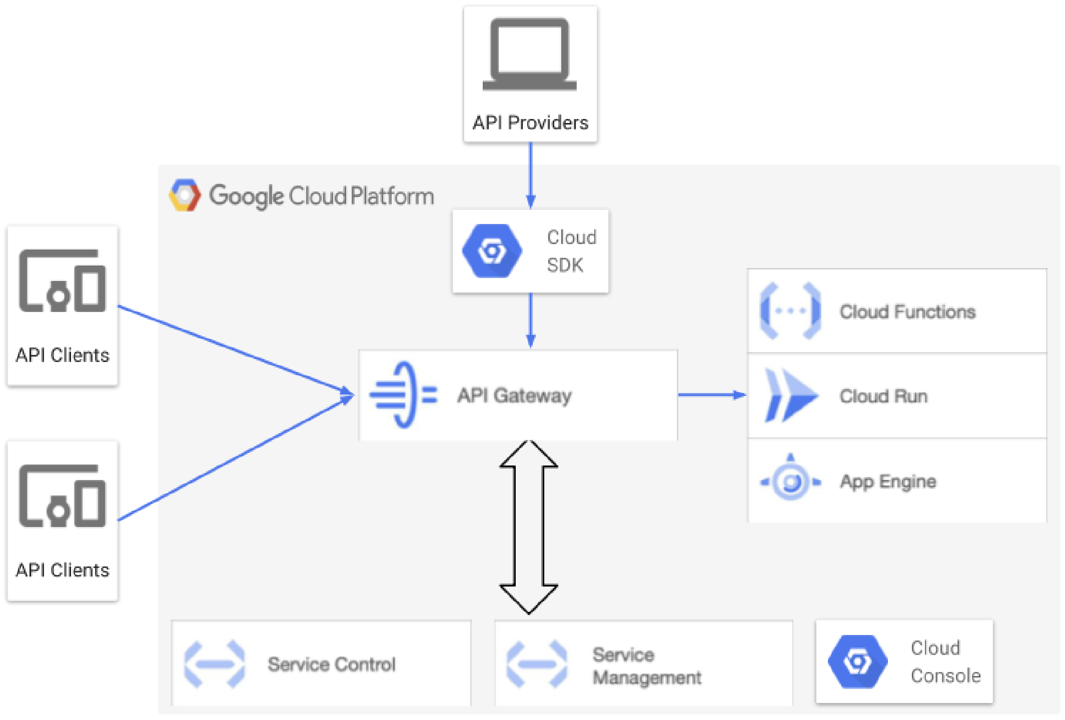
Homework 4: Gateways
Member: Abdul Bhutta
Changes: RED

The idea of gateway is central to a distributed systems. Please read the following articles and try to come with a descriptions for. The requirements of a gateway, and a high level design of the various components needed for a gateway to operate.

Gateway Components

Components	Description
Gateway	Managing the services
Control	Applying management rules
Management	Managing and configuration
Console	Deploy, manage, and monitor

An example of a high-level design or architecture for a gateway is shown below, which is used to implement a Google API Gateway. The main components are the gateway API, service control, service management, google cloud CLI, and cloud console.



Gateway Requirements

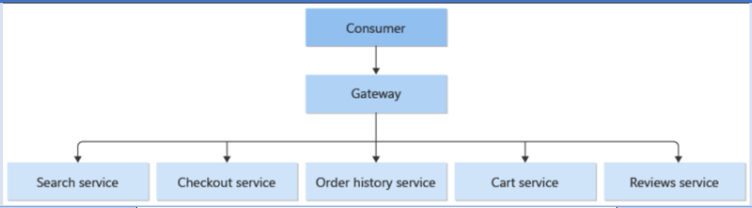
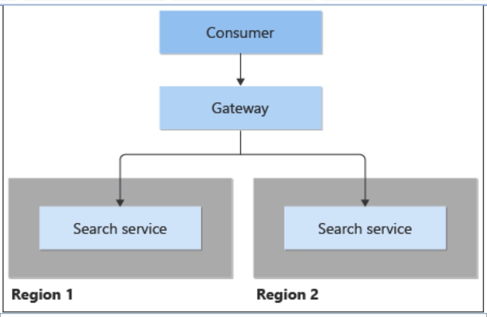
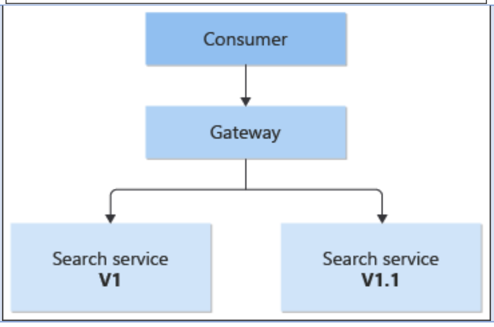
Requirement	Description
Frontend IP	Ip address to access the gateway
Port	To identify and listen to the request
Certificate	Provides security and privacy
Listeners	Used to listen to the request and logically send the request to the backend services.
Protocols	A type of communication among the components. (HTTP, HTTPS)

Requirements for Gateways
● *Reliable*: Gateway needs to be reliable, sending requests to the proper services, and being able to function correctly with heavy traffic loads. ● *Scalable*: Gateways can be scaled up and down with the traffic coming in, making for a more efficient system. ● *Availability*: Gateway should have a high availability to ensure clients can use the system at all times. ● *Performance*: The gateways should perform their tasks (Routing, Aggregation, Offloading) in an acceptable time. ● *Life-Time Costs*: The implementation of the gateway should have a lower cost when it is working for the system. ● *Reusability*: Reusability should be high for the implementation of the gateways, as this can be used in many other systems, reducing design and implementation times for future systems. ● *Maintainability*: The gateway's maintainability should be high, as in the case of a failure in a gateway it should be easy to mend the problem and get it back up and running quickly.

Gateway	**Description**
Gateway Routing	Route the various requests coming into various services into a single endpoint.
Gateway Offloading	Push shared service functionalities to a gateway proxy. Where it moves shared functionalities from applications to the gateway.
Gateway Aggregation	The gateway combines multiple incoming requests into a single request. Better when the client makes many requests to various backend systems.

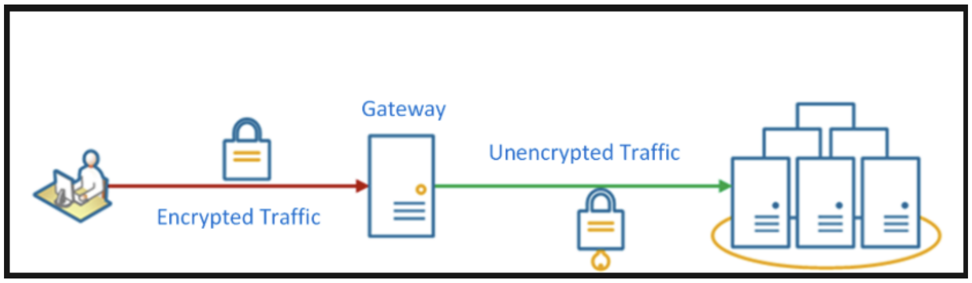
Gateway Routing Pattern

The concern is raised when multiple services, multiple instances of the same service, or multiple versions of the same service are updated, the client will be required to be updated as well. The solution to these concerns is deploying a gateway routing pattern which places a gateway at the endpoint and uses the application layer to route the traffic. This pattern allows the client to communicate with a single endpoint and hides the database or backend infrastructure. Examples of gateway routing patterns integrated within the applications are provided below.

Scenarios	Gateway Implentation
Multiple Services	 Consumer Gateway Search service Checkout service Order history service Cart service Reviews service
Multiple Instances of the the same servces	 Consumer Gateway Search service Search service Region 1 Region 2
Multiple Versions	 Consumer Gateway Search service V1 Search service V1.1

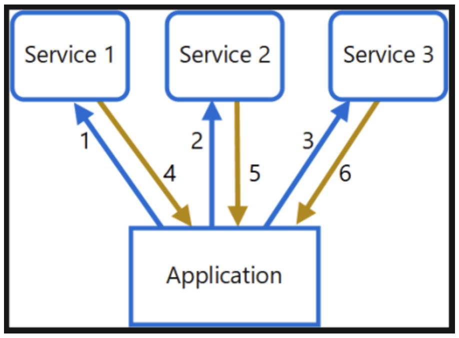
Gateway Offloading Pattern

As there are many applications which require services to be shared and deploying a service in each application causes overhead or deployments error. Some examples of services are security, authentication, and monitoring. Implementing each service with the application is complex, and the gateway offloading pattern is used to solve this issue. The solution is to add the services within the gateway and allowing to simplify the deployments. An example of a gateway offloading pattern is shown below.

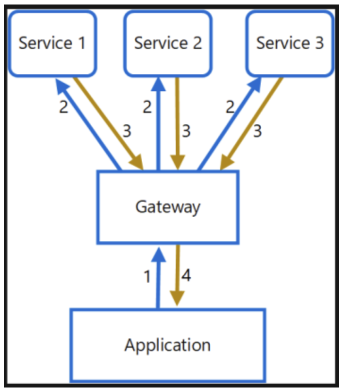


Gateway Aggregation Pattern

The gateway aggregation pattern is applied within applications which require communication with multiple requests to different databases or backend systems. When the client is sending multiple requests to the database or when it requires many services to perform one functionality, it will require the use of many resources, which will impact the performance and scalability of the application. An example of provided below where the application is requesting 3 services over a cellular network. The cellular network is unreliable, inefficient and has high latency, which may result in failure due to connectivity issues.



The solution is to implement a gateway in the middle to reduce the communication between the application and backend services. In the figure below, the gateway will receive the requests from the application, and the gateway will forward the requests to the services. The services will send back a response, and the gateway will combine the responses into a single response and forward it to the application.



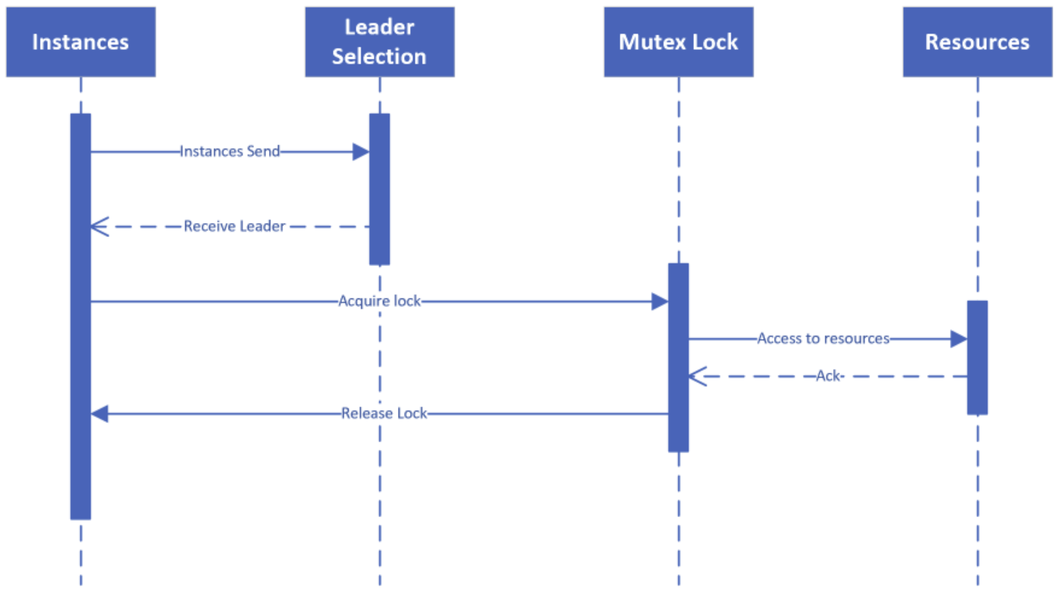
Because gateways are of such importance, they are typically using a clustering technique, where multiple instances of the gateways are deployed. Why would you do that?

The reason why you would have multiple instances that insinuate a clustering technique is because you can enable many high-availability clusters of gateway installations, that will help the client and the services use data resources from different cloud services that can help. With these cloud services (eg. Power BI), because gateways are so important, clusters are used to avoid any singular failure point when accessing any on-premise data resource. The gateway cloud service will be used in the primary gateway in a cluster unless the gateway isn't able to be used, which then allows the service to access the next available gateway in the cluster. Gateways use a clustering technique to allow redundancy, load balancing, and failover. This allows the use of another gateway in case one fails. The incoming requests are forwarded to the main gateway of the cluster, allowing for better performance and scalability.

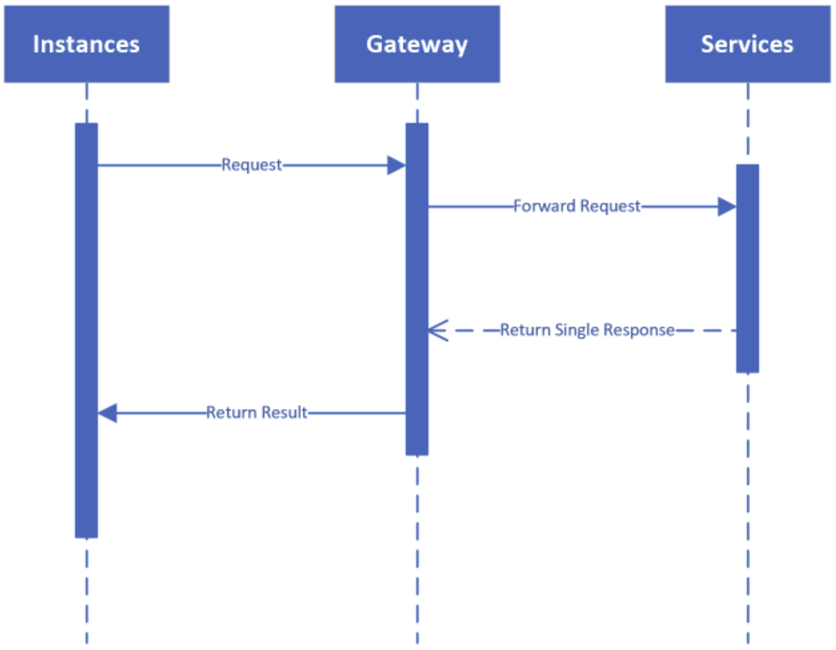
Leader Election Pattern

A typical cloud application consists of many tasks which may use the same resources in which a distributed system elects a leader to manger the other services. In a cloud application where multiple instances are provided to different users may be performing a task and needs to write to a database which is being shared by all the users. This may cause an issue as one task may overwrite the previous data. The leader election pattern is applied to coordinate and manage the instances.

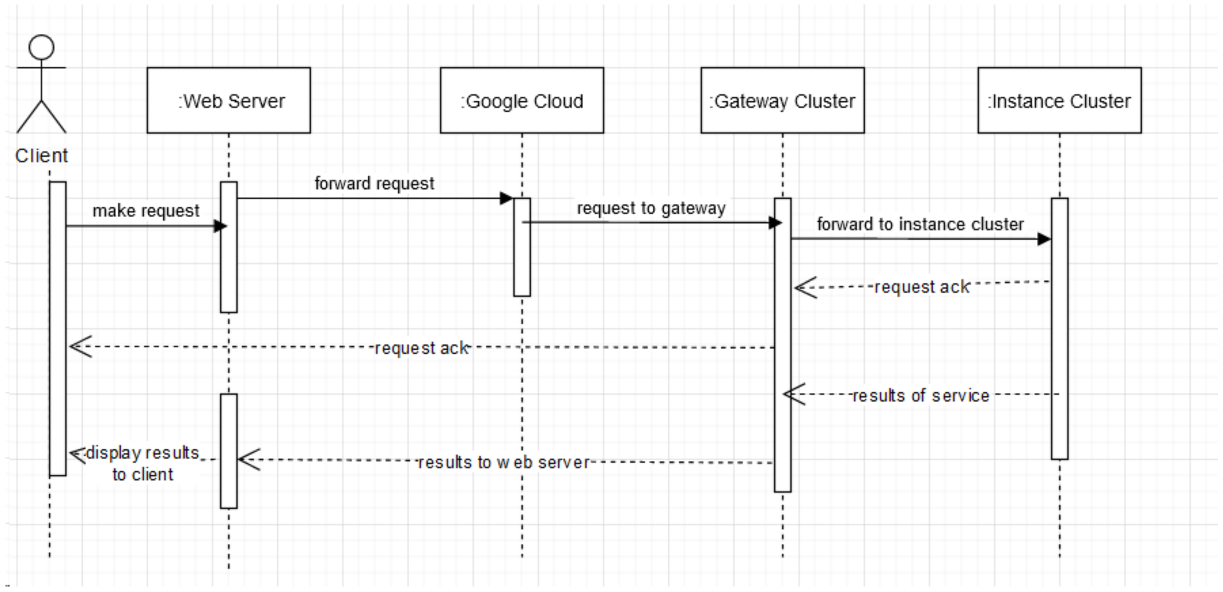
Sequence Diagram for Leader Election Algorithm



Sequence Diagram for Gateway and multiple Instances



Not sure which one is right



Homework 4: Gateways Member: Owen Musselman

Note: changes are in Red.

Part I:

Come up with descriptions for; The requirements of a gateway, and a high level design of the various components needed for a gateway to operate.

Gateway	Description
Gateway Routing	Route the various requests coming into various services into a single endpoint.
Gateway Offloading	Push shared service functionalities to a gateway proxy. Where it moves shared functionalities from applications to the gateway.
Gateway Aggregation	The gateway combines multiple incoming requests into a single request. Better when the client makes many requests to various backend systems.

Requirements for Gateways
● *Reliable*: Gateway needs to be reliable, sending requests to the proper services, and being able to function correctly with heavy traffic loads. ● *Scalable*: Gateways can be scaled up and down with the traffic coming in, making for a more efficient system. ● *Availability*: Gateway should have a high availability to ensure clients can use the system at all times. ● *Performance*: The gateways should perform their tasks (Routing, Aggregation, Offloading) in an acceptable time. ● *Life-Time Costs*: The implementation of the gateway should have a lower cost when it is working for the system. ● *Reusability*: Reusability should be high for the implementation of the gateways, as this can be used in many other systems, reducing design and implementation times for future systems. ● *Maintainability*: The gateway's maintainability should be high, as in the case of a failure in a gateway it should be easy to mend the problem and get it back up and running quickly.

Need to use application layer 7 so that requests are routed to the correct instance to be dealt with, this is gateway routing. With gateway routing there can be multiple disparate services, where the client can utilize various services, where the client is able to keep making requests, and the gateway will change its routing, allowing the client to not have to change anything. With gateway routing, there can be various instances of the same service, where the services connected to the gateway can be increased or decreased depending on the traffic to save resources. There can also be multiple versions of the same service, where the gateway can route the requests to a service that can serve them (one version can do something but another cannot)Can use something like nginx that will move SSL operations to one of various HTTP servers to be dealt with accordingly, this is gateway offloading. In order to implement this, gateways should have the gateway close to the physical location of the backend services in an attempt to reduce latency. Design should also be adequate and have gateway performance with meet growth and usage. Aggregation should be built into the gateway as it could have an effect on the routing and offloading of the gateway, and this will mitigate any inconsistencies in the offloading and routing. Through the use of aggregation it will decrease the amount of requests from the application to the services in the backend. Asynchronous input and output should be used to make sure there are small performance issues for the application. Additionally, it should have a robust design utilizing circuit breakers, timeouts, and bulkheads to make the system more available, and reliable. The design of gateways could have a single point of failure, and the gateway should be designed to accommodate the availability of the system.

Part II:

Because gateways are of such importance, they are typically using a clustering technique, where multiple instances of the gateways are deployed. Why would you do that? A technique to get the cluster to operate is to use the Leader Election pattern.

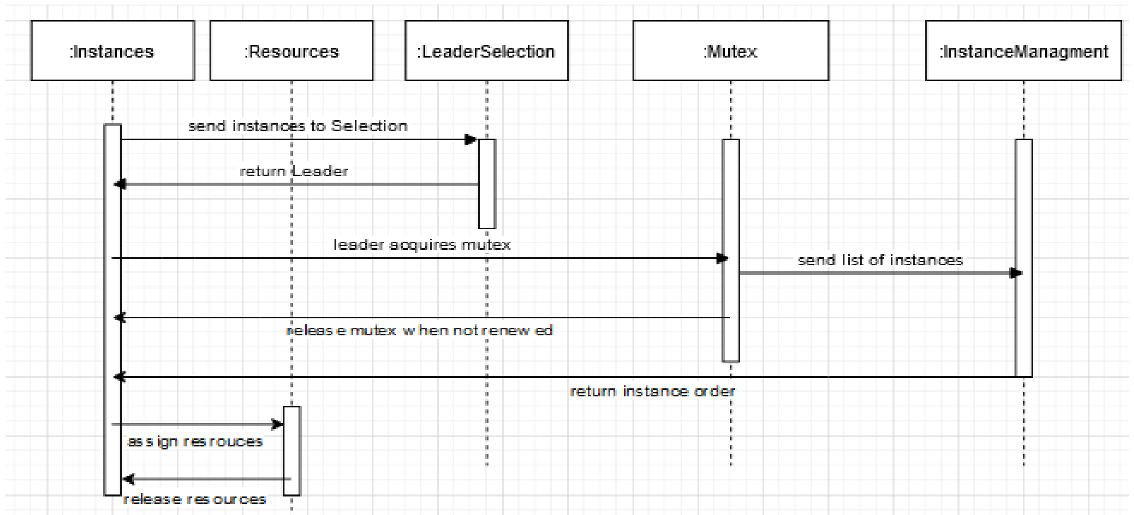
The use of clustering a gateway is an important one. By clustering gateways incoming requests are sent to the main gateway instance within the cluster. In the event that this main gateway is not operational, the incoming requests will be sent to another gateway within the cluster. This is accomplished through patterns like the Leader Election pattern, where the main component (gateway) is chosen. With the use of clustering and gateways it is possible to remove a single point of failure. The use of clustering thereby increases the availability, reliability and scalability.

Can you draw a UML sequence diagrams for the leader election algorithm? and a sequence diagram for gateway use cases using clusters of instances?

Leader Election Pattern:

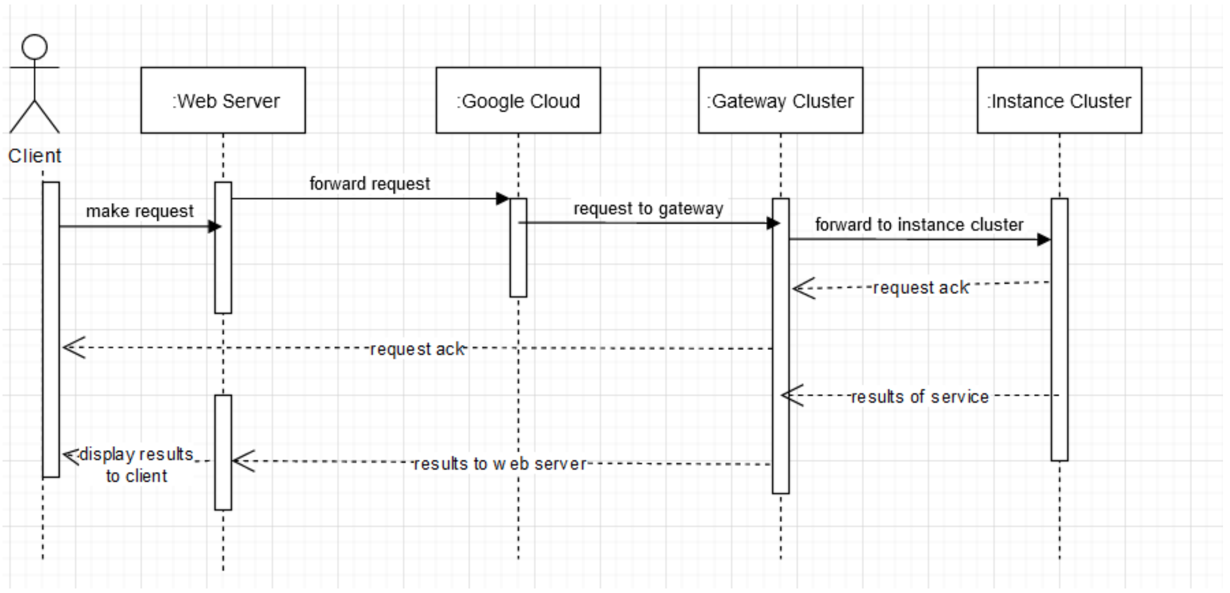
The Leader Election pattern harmonizes the operations that are done by a set of instances that are working together in a distributed application. This pattern will elect one of the instances of the distributed application as the leader, in which the leader is in charge of managing the other instances. This makes it so that the instances will not conflict with one another, like fighting over resources or other instance's work. If all instances are executing the same code then any of the instances can be selected as the leader. There are some drawbacks of this pattern, being that the leader could be unavailable at a certain time, or being able to sense that the leader is malfunctioning/failed. Another difficulty is that the leader election pattern needs to be robust, so that it can still function if there are failures.

UML Sequence Diagram - Leader Election



The set of instances is sent to a leader selection algorithm (could be the ring algorithm) where the leader is decided. Once decided that instance will acquire the mutex that belongs to the leader. When the mutex is acquired, the instances are sent to an Instance Management method where the instances are arranged in a way where they do not conflict with each other or shared resources, where the proper instances are assigned the resources, and then release them when they are done.

UML Sequence Diagram - Gateway use cases using clusters of instances



The client sends its request to the web server. This web server utilizes google cloud platform for clustering capabilities. The google cloud platform is just depicted to show that is what is being used in this case, but requests would be forwarded to the clusters. The client’s request is then forwarded to the gateway cluster where a main gateway has already been chosen using a pattern like the Leader Election pattern. From the gateway cluster the request is sent to the instance cluster where the request is processed. When the request reaches the instance cluster in this case, an acknowledgement is sent back to the client. When the request is served, the results are then sent back to the web server where it is displayed to the client. With the use of these clusters the availability, reliability, scalability, and performance are increased for example.

Homework 4: Gateways
Member: Alexander Campbell

Clustering by Consensus

Consensus Clustering is a technique in which a consensus is gained from various iterations of a particular data set. This is very important in distributed systems as there are often many copies of the same data set, doing consensus allows us to aggregate these in an accurate way. This technique of aggregation lends itself to several types of transparency. This supports replication transparency as it allows for there to be several copies of the same data without as many conflicts and without the user being aware. Due to having consensus several copies of data, this also leads to failure transparency where even if there is a failure in one point we can still aggregate data from the other functioning sources.

Leader Election Pattern

The Leader Election Pattern is a method to reduce conflict and issues with shared resources in a distributed system through the election of one deployed instance as the leader. When there is a leader instance to coordinate the actions of other deployments, we can avoid issues such as writing data to the same entry at the same time. The main difficulty in implementing this pattern is the required leader election system must be robust and still work despite any failures. This pattern supports concurrency transparency as it aids in several processes being able to run concurrently across a distributed system without causing conflict. **Infinispan**

Infinispan is an open source platform that utilizes data grids to store information. This system allows for distributed systems to store and retrieve data in an extremely performant manner. This means that the user will also be able to access services faster and improve the overall performance of the distributed system as a whole.

Gateway Routing Patterns

A Gateway Routing Pattern is when a distributed system utilizes a gateway interface so that traffic to several disparate services can be routed to one endpoint that will reroute or delegate to the multiple services. This is useful as clients will only need to access a single endpoint but be exposed to multiple services, without having to know their specific endpoints. Gateways support several types of transparencies including failure, performance, and scaling transparency. When using a gateway the control of traffic can be fine tuned, meaning we can redirect traffic to working deployments, direct them to underutilized resources, and know when scaling up or down is necessary as the gateway can measure demand. The major issue with this pattern is that it also may introduce a single point of failure within the distributed system, so the utilized gateway must be extremely robust. Outside of routing there are other ways to use specialized gateways to improve the performance of a distributed system.

Gateways can also be used in the Gateway Aggregation Pattern to increase performance in busy networks. In this pattern, we use a gateway to reduce the amount of separate backend calls to various services an application supports by aggregating multiple calls into one through the gateway. Gateways can also be configured to do administrative tasks in what is called the Gateway Offloading Pattern. Some features that involve gateways of some sort such as authentication, decryption, throttling, etc. can be offloaded to the gateway to make development of the application the gateway points to simpler and improve maintainability as a whole.

Homework 4: Gateways
Member: Mamun Hossain

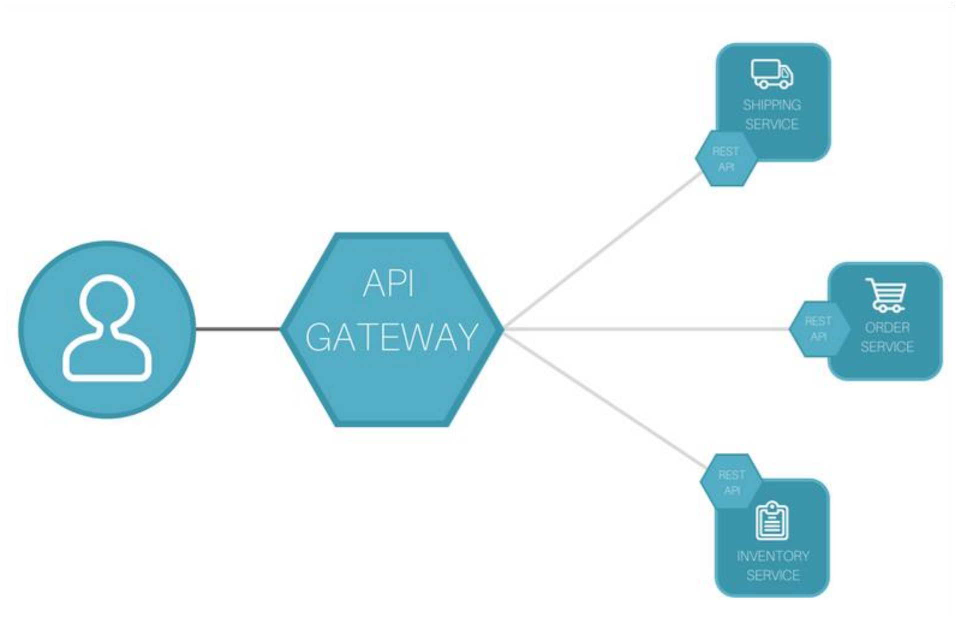
The idea of gateway is central to a distributed systems. Please read the following articles and try to come with a descriptions for;

The requirements of a gateway, and a high level design of the various components needed for a gateway to operate.

The goal of a gateway is to send requests to the backend server and then send that request back to the client. This is very crucial for a distributed system. Let's consider the gateway routing pattern, and think about the problem at hand: a client would need to consume different services, multiple, or both at once. We can also consider the gateway offloading pattern which has things such as services that would require configuration, management, and maintenance. For example, being able to handle the security issues at hand, says ass SSL certificate management. Even the gateway aggregation pattern uses a gateway to aggregate multiple individual requests to turn it into a single request, but how can that all be done?

Let us begin with the **gateway routing pattern**: The main requirement is having a client to have the need to consume different services, and gateway routing pattern ensures a way to have different endpoints handle the request and routes, as we can see in the image below.

Figure 1: High level Gateway Routing Pattern



The solution to this would be to simply apply a gateway to the front of the application or services that are being used. When we consider the OSI layers, we would would Layer 7, the application layer, to route the requests to the required instances. With this, the client will only need to know about the single endpoint they interact with.

Now consider the **gateway offloading pattern**: the requirements that are required for this pattern would be that the gateway must be highly available and cannot be easily prone to failure. You will also want to ensure that the gateway is scalable for the aforementioned requirements and the application and its endpoints, and you do not want the gateway to become a bottleneck for the application. Below is an image that depicts this.

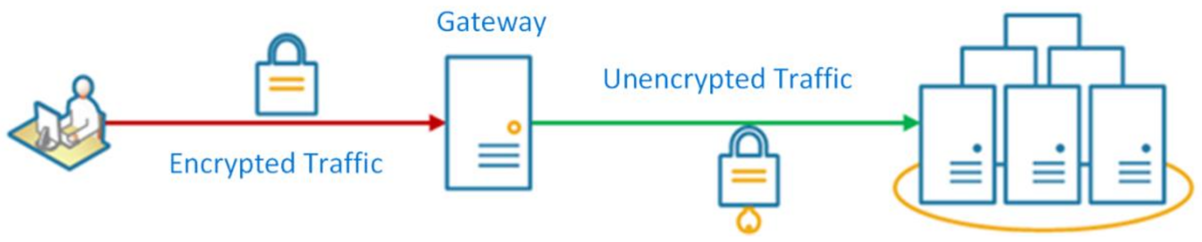


Figure 2: High level Gateway Offloading Pattern

You will want to use this pattern when a deployment has a concern with security, for example, a concern over the SSL certifications for the endpoints. If you use this pattern to track transactions, it would be highly beneficial to generating correlation IDs for logging purposes. That being said, this pattern is not always acceptable, for example it would not be suitable if there is coupling across different services.

Now for the **gateway aggregation pattern:** it is similar to the requirements of the gateway routing pattern, however it is also adding aggregation of services. What this means is that it offers a gateway service that aggregates multiple internal requests to different services and/or applications but it only exposes a single request to the client. So the requirements are the same as the routing, with the extra step. Below are high level designs that describe the way this gateway would operate.

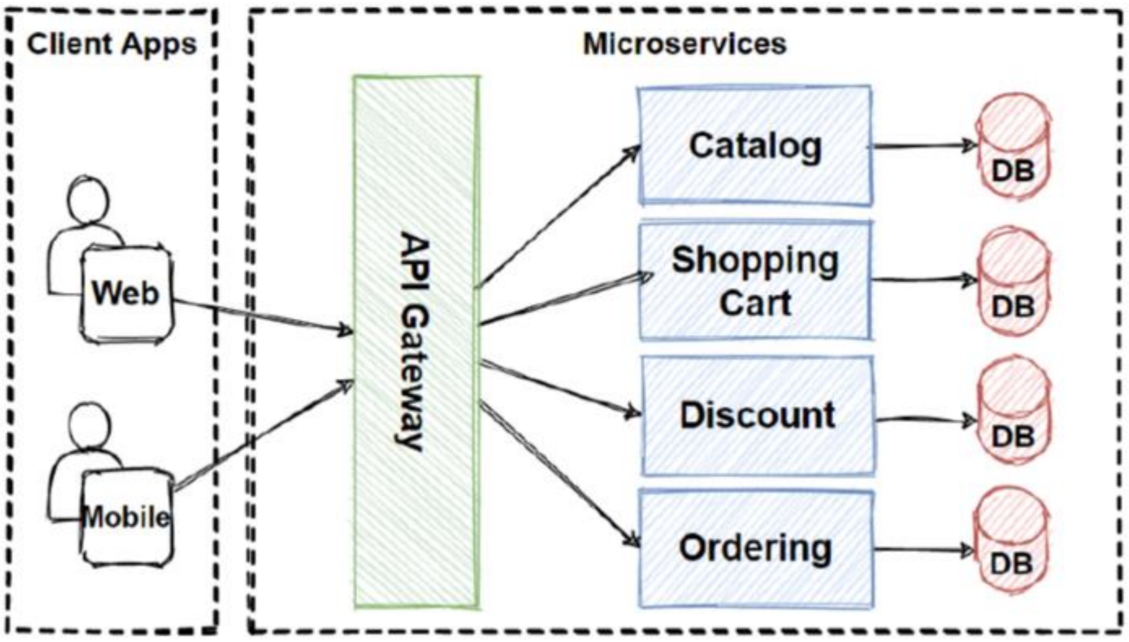


Figure 3 - Multiple clients

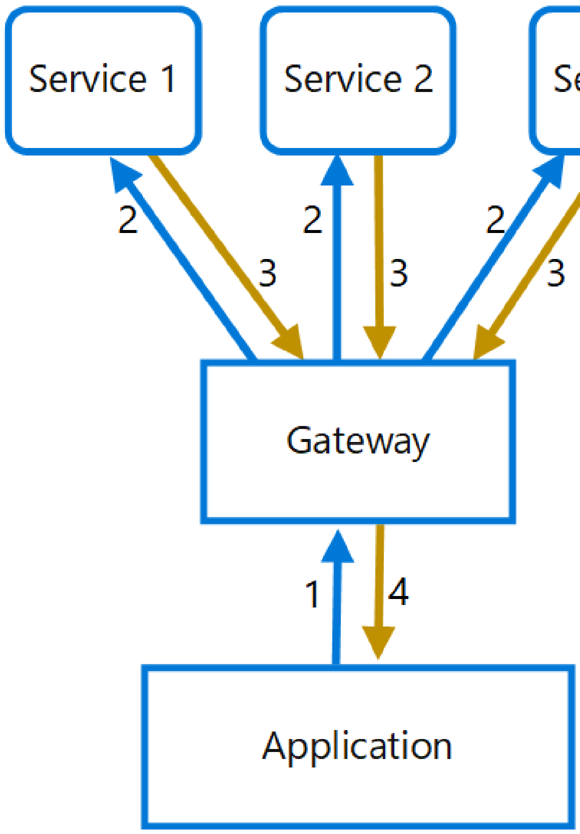


Figure 4: Single Application

The solution for this pattern is to include a gateway that reduces the chattiness between the client and the service. This way, the gateway will be able to retrieve the client requests individually, but will also be able to send the requests to different backend servers and then will aggregate the result that, which in turn, ends up sending the request back to the client. This also helps improve application performance over high-latency networks.

Because gateways are of such importance, they are typically using a clustering technique, where multiple instances of the gateways are deployed. Why would you do that?

The reason why you would multiple instances that insinuate a clustering technique is because you can enable many high-availability clusters of gateway installations, that will help the client and the services use data resources from different cloud services that can help. With these cloud services (eg. Power BI), because gateways are so important, clusters are used to avoid any singular failure point when accessing any on-premise data resource. The gateway cloud service will be used in the primary gateway in a cluster unless the gateway isn’t able to be used, which then allows the service to access the next available gateway in the cluster.

A technique to get the cluster to operate is to use the Leader Election Pattern:

To describe this technique is that it is used as a method to ensure that the shared resources in a distributed system have as minimal conflict as possible. The way this works is through an election of task instances that is picked to be the leader, and the leader will then coordinate the other instances in the protocol that needs to be administered for that system. However, if all the task instances are running the same code, then they are all capable of being the leader.

With this leader election pattern, we can be sure that we can avoid problems such as writing data to the same entry at the same time. However, some issues that arise are the following: being able to detect when the leader has failed, or just unavailable. The main difficulty with this pattern is that the leader must be robust and must work without any failures. We would tend to use this pattern to avoid making a leader a bottleneck in the system so the work can be coordinated efficiently.

Can you draw UML sequence diagrams for the leader election algorithm? and a sequence diagram for gateway use cases using clusters of instances?

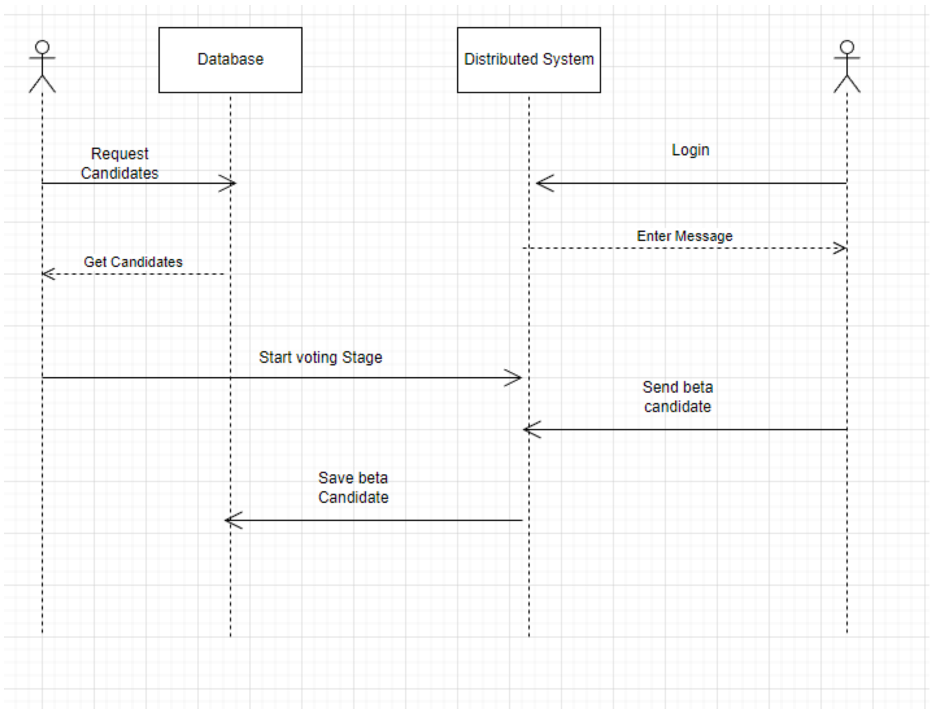


Figure 5: Leader Election Algorithm sequence diagram

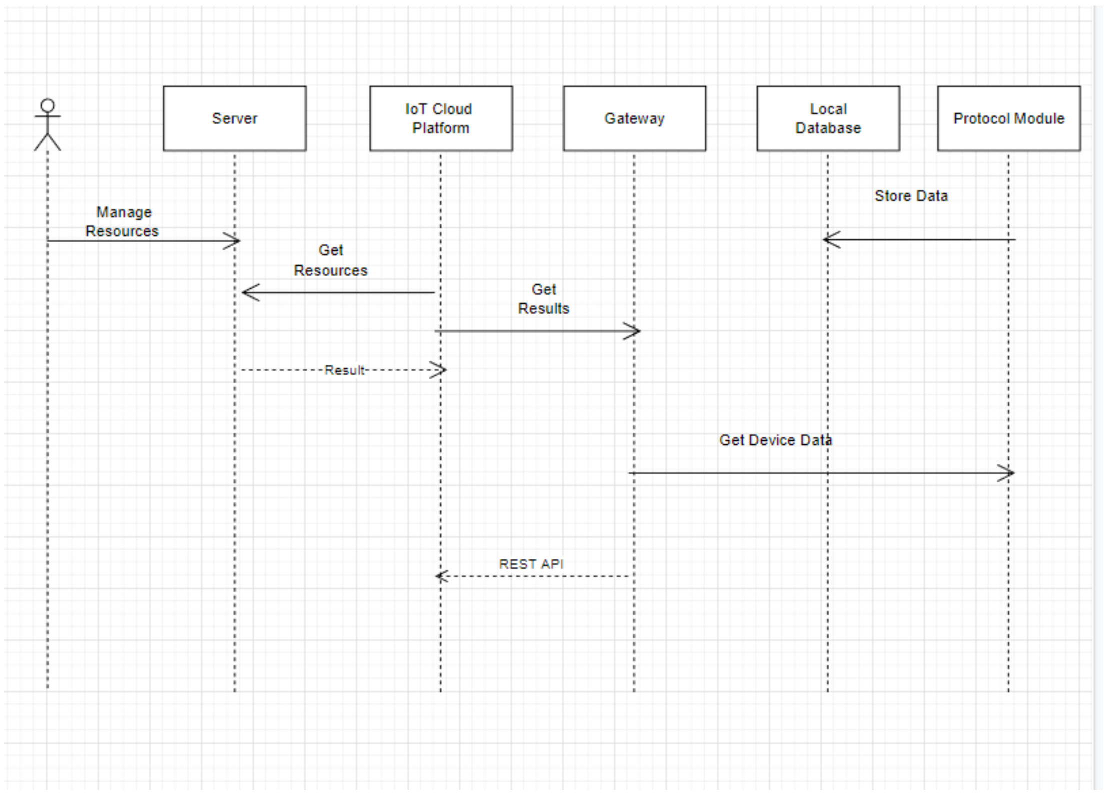


Figure 6: Sequence Diagram for Gateway Use Cases