1. **What does an API gateway do? What is the implementation of the API gateways?**

An API gateway is a server that acts as an entry point for a group of microservices. It is responsible for receiving and routing API requests from clients, as well as providing services such as rate limiting, authentication and authorization, traffic management, and API monitoring.

The implementation of an API gateway can vary depending on the specific requirements of the system it serves. However, typically an API gateway is implemented as a reverse proxy server, sitting in front of a set of microservices. The API gateway receives incoming requests from clients, and based on the request URL, headers, or other parameters, routes the request to the appropriate microservice. The API gateway may also perform additional tasks such as protocol translation, request/response transformation, and caching to improve performance.

API gateways can be implemented using a variety of technologies such as Nginx, Apache, Kong, Tyk, and AWS API Gateway. They can also be implemented as part of a larger API management platform that provides additional capabilities such as developer portal, analytics, and monetization.

1. **Compare cloud-based gateways with the self-hosted ones, when choose either of them?**

Cloud-based API gateways and self-hosted API gateways have their own advantages and disadvantages, and the choice between them depends on the specific needs of the organization.

Cloud-based API gateways are hosted and managed by a third-party provider and are accessible through the internet. They offer the following benefits:

1. Scalability: Cloud-based API gateways are highly scalable and can handle many API requests.
2. Availability: Cloud-based API gateways offer high availability and uptime, as they are hosted on cloud infrastructure that is designed for fault-tolerance.
3. Maintenance: Cloud-based API gateways require less maintenance, as the provider takes care of most of the tasks related to infrastructure management.
4. Cost: Cloud-based API gateways are cost-effective, as the provider charges a subscription fee based on usage, and there are no upfront costs.

On the other hand, self-hosted API gateways are installed and managed on the organization's own infrastructure. They offer the following benefits:

1. Control: Self-hosted API gateways give the organization full control over the infrastructure and the ability to customize the gateway to meet specific needs.
2. Security: Self-hosted API gateways offer better security, as the organization has complete control over the infrastructure and can implement its own security policies.
3. Integration: Self-hosted API gateways can be easily integrated with the organization's existing infrastructure and systems.
4. Cost: While self-hosted API gateways require upfront costs for infrastructure and maintenance, they may be more cost-effective in the long run for organizations that have many API requests.

In general, organizations that have limited resources or are looking for a quick and easy solution may prefer cloud-based API gateways. Organizations that require more control, security, and customization may opt for self-hosted API gateways.

1. **What is Backend for Frontend (BFF) pattern? What are the pros and cons of it?**

The Backend for Frontend (BFF) pattern is an architectural pattern that proposes creating a dedicated backend service for each client application or frontend, instead of having a single backend service that serves all clients. This approach allows for a more tailored backend service that meets the specific needs of each client, rather than forcing the frontend to work around the limitations of a generic backend.

Some of the advantages of the BFF pattern include:

* Improved developer productivity: Developers can work on the backend and frontend in parallel, which can improve development speed and reduce dependencies between teams.
* Improved performance: With a dedicated backend for each frontend, it's easier to optimize the backend to meet the specific needs of each frontend. This can result in better performance and a better user experience.
* Improved security: With a dedicated backend for each frontend, it's easier to apply specific security policies and controls to each frontend, rather than having a generic set of policies that may not be sufficient for all clients.

Some of the disadvantages of the BFF pattern include:

* Increased complexity: With multiple backend services to manage, there's a risk of increased complexity and management overhead.
* Increased cost: With multiple backend services, there may be additional infrastructure and resource costs associated with managing and maintaining those services.
* Potential for inconsistent APIs: With multiple backend services, there's a risk that the APIs exposed to clients may be inconsistent, which can create confusion and make it more difficult for clients to integrate with the backend.

Overall, the decision to use the BFF pattern depends on the specific needs of the application and the trade-offs between developer productivity, performance, security, complexity, and cost.

1. **How can you degrade or improve API Gateway performance?**

There are several ways to improve or degrade the performance of an API Gateway. Here are a few:

* Caching: Caching can improve the performance of an API Gateway by reducing the number of requests sent to the backend services. The Gateway can cache the responses to requests, and the subsequent requests for the same data can be served from the cache instead of the backend.
* Load balancing: Load balancing can distribute the incoming traffic across multiple servers to improve the overall performance and availability of the API Gateway.
* Compression: Compressing the responses can reduce the size of the data sent between the API Gateway and clients, resulting in faster response times.
* Routing rules: Defining routing rules that limit the number of API calls to specific services can improve the performance of an API Gateway by reducing the workload of the backend services.
* Monitoring: Regularly monitoring the performance of the API Gateway can help identify bottlenecks and other performance issues, which can then be addressed to improve the overall performance.

On the other hand, the following factors can degrade the performance of an API Gateway:

* Too many calls: When there are too many API calls, it can cause the Gateway to become overloaded and slow down.
* Poorly optimized code: If the code running on the API Gateway is not optimized, it can result in slow response times.
* Inadequate hardware resources: If the API Gateway does not have adequate hardware resources, it can affect its performance.
* Network latency: High network latency can cause delays in the communication between the API Gateway and backend services, resulting in slower response times.
* Security checks: If the API Gateway has to perform multiple security checks, it can slow down the processing of API requests.