1. **What are microservices? What are the benefits of this architectural style?**

**Response:**

[Microservices](https://pivotal.io/microservices) is an architectural approach to developing an application as a collection of small services; there are certain common characteristics around organization around business capability, automated deployment, intelligence in the endpoints, and decentralized control of languages and data.

Each microservice can be deployed, upgraded, scaled, and restarted independent of other services in the application, typically as part of an automated system, enabling frequent updates to live applications without impacting end customers.

 Microservices architecture decomposes applications into small, loosely coupled independently operating services. These services map to smaller, independent development teams and make possible frequent, independent updates, scaling, and failover/restart without impacting other services.

1. **What are the limitations of traditional configuration? What are the advantages of externalizing configuration settings?**

**Response:**

**Limitations of Traditional Configuration**

1. **Unpredictable.** Traditional configuration applications can’t realize all of the benefits of running on a cloud-native platform due to the unique way each one is architected or developed. This type of application often takes longer to build, is released in big batches, can only scale gradually, and assumes high availability of dependent services.
2. **OS dependent.** Traditional configuration application architecture allows developers to build close dependencies between the application and underlying OS, hardware, storage, and backing services. These dependencies make migrating and scaling the application across new infrastructure complex and risky, working against the cloud model.
3. **Over-sized capacity.** Traditional IT designs a dedicated, custom infrastructure solution (“snowflake”) for an application, delaying deployment of the application. The solution is often over-sized based on worst-case capacity estimates with little capability to scale beyond to meet demand.
4. **Siloed.** Traditional configuration operates an over-the-wall handoff of finished application code from developers to operations, which then runs it in production. Organizational priorities take precedence over customer value, resulting in internal conflict, slow and compromised delivery, and poor staff morale.
5. **Waterfall development**. IT teams release software periodically, typically weeks or months apart, when code has been built into a release despite the fact that many of the components of the release are ready earlier and have no dependency other than the artificial release vehicle. Features that customers want or need are delayed and the business misses opportunities to compete, win customers, and grow revenue.
6. **Dependent.** It bundles many disparate services into a single deployment package causing unnecessary dependencies between services and leading to a loss of agility during development and deployment.
7. **Manual scaling.** It includes human operators that manually craft and manage server, network, and storage configurations. At scale, operators are slow to correctly diagnose issues and easily fail to correctly implement at scale due to the level of complexity. Hand-crafted automation recipes have the potential to hard-code human errors into the infrastructure.
8. **Slow recovery.** Traditional configuration is a slow and inefficient foundation for microservice-based applications because individual VMs are slow to startup/shutdown and come with large overhead even before deploying application code to them.

**Advantages of externalized configuration**

externalized configuration is a pattern that allows an application to run in multiple environments (e.g. development, test, production) without any modifications in the application itself.

In microservices architectures, systems are split into several services (microservices), each one usually running in a separate process. Each process can be deployed and scaled independently, and this means there may be many instances of the same microservice running at a certain time.

Let’s say you want to modify the configuration for a microservice that has been replicated a hundred times (one hundred processes are running). If the configuration for this microservice is packaged with the microservice itself, you’ll have to redeploy each of the one hundred instances. This can result in some instances using the old configuration, and some using the new one, at some point. Moreover, sometimes microservices use external connections which, for example, require URLs, usernames, and passwords. If you want to update these settings, it would be useful to have this configuration shared across services.

Externalized configuration works by keeping the configuration information in an external store, such as a database, file system, or environment variables. At startup, microservices load the configuration from the external store. During runtime, microservices provide an option to reload the configuration without having to restart the service.

1. **What is Spring Cloud? List and explain five features.**

**Response:**

Spring Cloud provides tools for developers to quickly build some of the common patterns in distributed systems (e.g. configuration management, service discovery, circuit breakers, intelligent routing, micro-proxy, control bus, one-time tokens, global locks, leadership election, distributed sessions, cluster state). Coordination of distributed systems leads to boiler plate patterns and using Spring Cloud developers can quickly stand up services and applications that implement those patterns. They will work well in any distributed environment, including the developer’s own laptop, bare metal data centers, and managed platforms such as Cloud Foundry.

Features

Spring Cloud focuses on providing good out of box experience for typical use cases and extensibility mechanism to cover others.

**1. Service registration and discovery**

A service registry is a phone book for your microservices. Each service registers itself with the service registry and tells the registry where it lives (host, port, node name) and perhaps other service-specific metadata - things that other services can use to make informed decisions about it. Clients can ask questions about the service topology (“are there any ‘fulfillment-services’ available, and if so, where?”) and service capabilities (“can you handle X, Y, and Z?”). You probably already use a technology that has some notion of a cluster (Cassandra, Memcached, etc.), and that information is ideally stored in a service registry. There are several popular options for service registries. Netflix built and then open-sourced their own service registry, Eureka. Another new, but increasingly popular option is Consul. We’ll look principally at some of the integration between Spring Cloud and Netflix’s Eureka service registry.

**2. Routing**

Routing is an integral part of a microservice architecture. For example, / may be mapped to your web application, /api/users is mapped to the user service and /api/shop is mapped to the shop service. Zuul is a JVM-based router and server-side load balancer from Netflix.

Netflix uses Zuul for the following:

• Authentication

• Insights

• Stress Testing

• Canary Testing

• Dynamic Routing

• Service Migration

• Load Shedding

• Security

• Static Response handling

• Active/Active traffic management

Zuul’s rule engine lets rules and filters be written in essentially any JVM language, with built-in support for Java and Groovy.

**3. Load balancing**

Load balancing refers to efficiently distributing incoming network traffic across a group of backend servers, also known as a server farm or server pool.

A load balancer acts as the “traffic cop” sitting in front of your servers and routing client requests across all servers capable of fulfilling those requests in a manner that maximizes speed and capacity utilization and ensures that no one server is overworked, which could degrade performance. If a single server goes down, the load balancer redirects traffic to the remaining online servers. When a new server is added to the server group, the load balancer automatically starts to send requests to it.

In this manner, a load balancer performs the following functions:

• Distributes client requests or network load efficiently across multiple servers

• Ensures high availability and reliability by sending requests only to servers that are online

• Provides the flexibility to add or subtract servers as demand dictates

Load Balancing Algorithms

Different load balancing algorithms provide different benefits; the choice of load balancing method depends on your needs:

• Round Robin – Requests are distributed across the group of servers sequentially.

• Least Connections – A new request is sent to the server with the fewest current connections to clients. The relative computing capacity of each server is factored into determining which one has the least connections.

• IP Hash – The IP address of the client is used to determine which server receives the request.

**4. Circuit Breakers**

Microservices architecture has many advantages. These include low coupling, re-usability, business agility and distributed cloud ready applications. But at the same time, it makes the architecture brittle because each user action results invokes multiple services. It replaces the in-memory calls of a monolithic architecture with remote calls across the network, which works well when all services are up and running. But when one or more services are unavailable or exhibiting high latency, it results in a cascading failure across the enterprise. Service client retry logic only makes things worse for the service and can bring it down completely.

The Circuit breaker pattern helps to prevent such a catastrophic cascading failure across multiple systems. The circuit breaker pattern allows you to build a fault tolerant and resilient system that can survive gracefully when key services are either unavailable or have high latency.

Circuit breakers allow your system to handle these failures gracefully. The circuit breaker concept is straightforward. It wraps a function with a monitor that tracks failures. The circuit breaker has 3 distinct states, Closed, Open, and Half-Open:

1. Closed – When everything is normal, the circuit breaker remains in the closed state and all calls pass through to the services. When the number of failures exceeds a predetermined threshold the breaker trips, and it goes into the Open state.
2. Open – The circuit breaker returns an error for calls without executing the function.
3. Half-Open – After a timeout period, the circuit switches to a half-open state to test if the underlying problem still exists. If a single call fails in this half-open state, the breaker is once again tripped. If it succeeds, the circuit breaker resets back to the normal closed state.

**5. Leadership election and cluster state**

• In distributed computing, leader election is the process of designating a single process as the organizer of some task distributed among several computers (nodes). Before the task is begun, all network nodes are either unaware which node will serve as the "leader" (or coordinator) of the task, or unable to communicate with the current coordinator. After a leader election algorithm has been run, however, each node throughout the network recognizes a particular, unique node as the task leader.

• The network nodes communicate among themselves in order to decide which of them will get into the "leader" state. For that, they need some method in order to break the symmetry among them. For example, if each node has unique and comparable identities, then the nodes can compare their identities, and decide that the node with the highest identity is the leader.

• The definition of this problem is often attributed to LeLann, who formalized it as a method to create a new token in a token ring network in which the token has been lost.

• Leader election algorithms are designed to be economical in terms of total bytes transmitted, and time. The algorithm suggested by Gallager, Humblet, and Spira[1] for general undirected graphs has had a strong impact on the design of distributed algorithms in general, and won the Dijkstra Prize for an influential paper in distributed computing.

• Many other algorithms have been suggested for different kinds of network graphs, such as undirected rings, unidirectional rings, complete graphs, grids, directed Euler graphs, and others. A general method that decouples the issue of the graph family from the design of the leader election algorithm was suggested by Korach, Kutten, and Moran.