

**CHENNAI – PONDICHERRY**

**Consistent Disaster Recovery for Microservices: the BAC Theorem**

**Abstract:**

How do you back up a microservice? You dump its database. But how do you back up an entire application decomposed into microservices? In this article, we discuss the tradeoff between the availability and consistency of a microservice-based architecture when a backup of the entire application is being performed. We demonstrate that service designers have to select two out of three qualities: backup, availability, and/or consistency (BAC). Service designers must also consider how to deal with consequences such as broken links, orphan state, and missing state.

**Existing System:**

Micro services are about the design of fine-grained services, which can be developed and operated by independent teams, ensuring that an architecture can organically grow and rapidly evolve.1 By definition, each microservice is independently deployable and scalable; each stateful one relies on its own polyglot persistent storage mechanism. Integration at the database layer is not recommended, because it introduces coupling between the data representation internally used by multiple microservices. Instead, microservices should interact only through well-defined APIs, which—following the REST architectural style2—provide a clear mechanism for managing the state of the resources exposed by each microservice. Relationships between related entities are implemented using hypermedia,3 so that representations retrieved from one microservice API can include links to other entities found on other microservice APIs. While there is no guarantee that a link retrieved from one microservice will point to a valid URL served by another, a basic notion of consistency can be introduced for the microservice-based application, requiring that such references can always be resolved, thus avoiding broken links.

**Proposed System:**

*Broken link*: This is when the reference can no longer be followed—for example, where, using database terminology, foreign key records are recovered without the corresponding primary key records. In Figure 3, the up-to-date Microservice A refers to an obsolete version of the other Microservice B, where the referenced entity cannot be found after it has been recovered. The inconsistency can thus be detected when clients try to follow the reference from A to B and instead find that the link is broken.

• *Orphan state*. This is when there is no reference to be followed—for example, where primary key records are recovered without the corresponding foreign key records. In Figure 4, the state of the up-to-date Microservice A is no longer referenced from the state of the Microservice B recovered from an obsolete backup. This situation might be more difficult to detect from clients, because there are no immediately visible inconsistencies when they interact with either service. However, this might introduce storage space leaks for A, if there is no garbage-collection mechanism for such orphan states. Moreover, duplicate event log entries for A might be introduced (which might lead to unwanted side effects), as the obsolete Microservice B is brought up to date.