Algolab

12 Critical Strategies of System Design

Design For Failure

Any component that can fail will eventually fail, so it is important to design the service in a way that allows it to recover from failures gracefully.

The service should also be able to survive failures without the need for human intervention.

Redundancy And Fault Recovery

To ensure the reliability and robustness of the service, it is important to document all possible failure modes of each component, as well as combinations of those failure modes. Redundancy should also be built into any component that has the potential to fail. Additionally, it is important to be able to take down any server in the service without disrupting the workload.

Single-Version Software

Aim to have only one version of your software. This is simpler to manage and cost-effective.

It may be easier to achieve this for consumer-based services compared to enterprise software services.

For enterprise software services, try to use a single version or consider outsourcing the service to a host or application service provider.

Multi-tenancy

Aim to have multiple tenants within the same service.

This is more efficient and cost-effective compared to having separate services for each tenant, which can be more difficult to manage.

Quick Service Health Check

It is important for services to have a simple and fast health check endpoint. While it is not possible to test for all possible scenarios, a quick health check can help speed up the process of merging new code and quickly detect any failures.

Algolab

Develop In The Full Environment

It is important for development to happen in an environment that is similar to production. This way, developers can test all aspects of production scenarios on their own machines and ensure that their code will work properly in a live production environment.

Algolab

Zero Trust in Underlying Components

Expect that underlying components may fail at some point.

Some recovery techniques that can be used in these situations include operating in read-only mode using cached data, or continuing to provide service to most users while a redundant copy of the failed component is accessed.

Do Not Build the Same Functionality in Multiple Components

It is important to avoid code redundancy, similar to the DRY (Don't Repeat Yourself) principle.

If redundancy is allowed to creep into the code, fixes will need to be made in multiple parts of the system, which can be time-consuming and error-prone. Without proper care, the code base can quickly become difficult to maintain.

One Cluster Should Not Affect Another Cluster

Clusters should be as independent as possible, with minimally correlated failures.

Global services, even with redundancy, can be a central point of failure, so it is best to avoid them whenever possible. Instead, try to include everything that a cluster needs within the cluster itself. This can help to reduce the risk of failures and improve the reliability of the system.

Allow (rare) Emergency Human Intervention.

Design the system to operate without human intervention, but have a plan in place for rare events that may require human intervention due to combined or unanticipated failures.

Test troubleshooting steps as scripts in production to ensure they work and periodically conduct "fire drills" to test the readiness of the operations team.

Keep Things Simple And Robust

It is generally better to use simple and straightforward solutions instead of complicated algorithms and complex interactions between services.

As a general rule, it is worth considering optimizations that bring about an order of magnitude improvement, but smaller gains in performance, such as percentage or small factor improvements, may not be worth the additional complexity.

Enforce Admission Control At All Levels

A well-designed system should have admission control in place at the entry point to prevent overloading. This follows the principle that it is better to prevent more work from entering an already overloaded system rather than accepting more work and causing thrashing.

As a general rule, it is better to degrade gracefully rather than failing completely and providing poor service to all users. To avoid this, it is important to block access to the service before it becomes overwhelmed.

Follow Touseef on Linkedin to be notified of more such posts.