# What Causes Poor Availability?

What causes an application that previously performed well to begin exhibiting poor availability? There are many possible causes:

*Resource exhaustion*

Increase the number of users or increase the amount of data in use in a system and your application may fall victim to resource exhaustion, resulting in a slower and unresponsive application.

*Unplanned load-based changes*

Increases in the popularity of your application might require code and application changes to handle the increased load. These changes, often implemented quickly and at the last minute with little or no forethought or planning, increase the likelihood of problems occurring.

*Increased number of moving parts*

As an application gains popularity, it is often necessary to assign more and more developers, designers, testers, and other individuals to work on and maintain it. This larger number of individuals working on the application creates a large number of moving parts, whether those moving parts are new features, changed features, or just general application maintenance. The more individuals working on the application, the more moving parts within the application and the greater the chance for bad interactions to occur in it.

*Outside dependencies*

The more dependencies your application has on external resources, such as SaaS services, infrastructure, or cloud-based services, the more it is exposed to availability problems caused by those resources.

*Technical debt*

Increases in the applications complexity typically increase technical debt (i.e., the accumulation of desired software changes and pending bug fixes that often build up over time as an application grows and matures). Technical debt increases the likelihood of a problem occurring.

**Ways to improve Availability:**

Knowing what you can do when your availability begins to slip will help you to avoid falling into a vicious cycle of problems. What can you do to avoid your availability slipping? Some key things are:

* Measure and track your current availability
* Automate your manual processes
* Automate your deployment processes
* Maintain and track all configurations in a management system
* Allow quick changes and experiments, with an easy rollback capability if a problem occurs
* Aim to continuously improve your applications and systems
* Keep on top of availability as a core issue as your application changes and grows

# Five Focuses to Improve Application Availability

## Focus #1: Build with Failure in Mind

## Focus #2: Always Think About Scaling

## Focus #3: Mitigate Risk

## Focus #4: Monitor Availability

* Server monitoring
* Configuration change monitoring
* Application performance monitoring
* Synthetic testing
* Alerting

# Splitting into Services

A service is a standalone component. The word standalone is critical. A service meets the following criteria:

*Maintains its own code base*

A service has its own code base that is distinct from the rest of your code base.

*Manages its own data*

A service that requires maintaining state has its own data that is stored in its own data store. The only access to this separated data is via the service’s defined API. No other service may directly touch another service’s data or state information.

*Provides capabilities to others*

A service has a well-defined set of capabilities, and it provides these capabilities to other services in your application. In other words, it provides an API.

*Consumes capabilities from others*

A service uses a well-defined set of capabilities provided by others and uses them in a standard, supported manner. In other words, it uses other services’ APIs.

*Single owner*

A service is owned and maintained by a single development team within your organization. A single team may own and maintain more than one service, but a single service can have only one team that owns and maintains it.

# Dealing with Service Failures

* Predictable
* Understandable
* Reasonable for the situation

##### [Building Systems with Reduced Risk](https://learning.oreilly.com/library/view/architecting-for-scale/9781492057161/ch11.html)

* Introduce Redundancy
* Understand Independence
* Manage Security
* Encourage Simplicity
* Build in Self-Repair
* Standardize on Operational Processes

# Six Levels of Cloud Maturity

* Level 1—Experimenting: What is the cloud?
* Level 2—Securing the cloud: Can we trust the cloud?
* Level 3—Enabling servers and SaaS: Lift-and-shift, confirmation the cloud works well
* Level 4—Enabling value-added services: Dynamic cloud becomes a practice
* Level 5—Enabling unique services: Dynamic cloud is deeply ingrained in the culture
* Level 6—Mandating cloud usage: Why do we need our own data centres?